



PÓS-GRADUAÇÃO MIT ENGENHARIA DE SOFTWARE COM .NET

Projeto de Bloco: Engenharia de Software e Modelagem

Integrantes:

ALUNO: ALEXANDER SILVA

ALUNO: YURI ALVES

ALUNO: PEDRO NOVAES



Projeto de Bloco: Engenharia de Software e Modelagem



MODELAGEM ESTRATÉGICA

Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

<https://www.eduardopires.net.br/2016/08/ddd-nao-e-arquitetura-em-camadas>

Qual problema vai resolver?;
O que vai implementar?;
Para que serve?;
Quem vai atender?
Quando implementar o DDD?

Entender a fundo o negócio



Projeto de Bloco: Engenharia de Software e Modelagem

Pontos Chave

Ubiquitous Language

Uma linguagem onipresente, universal de um negócio dentro da empresa, compartilhada por todas as partes envolvidas no projeto, ou seja, termos específicos do domínio.

Aplicar os conceitos de DDD

Bounded Context

É uma parte da empresa ou do domínio do negócio que possui elementos ou conceitos com significados bem definidos, com linguagem própria, arquitetura e implementação específica.

Context Map

É um mapa que representa todos os contextos mapeados através dos elementos da linguagem ubíqua, bem como seus subdomínios e relacionamentos.

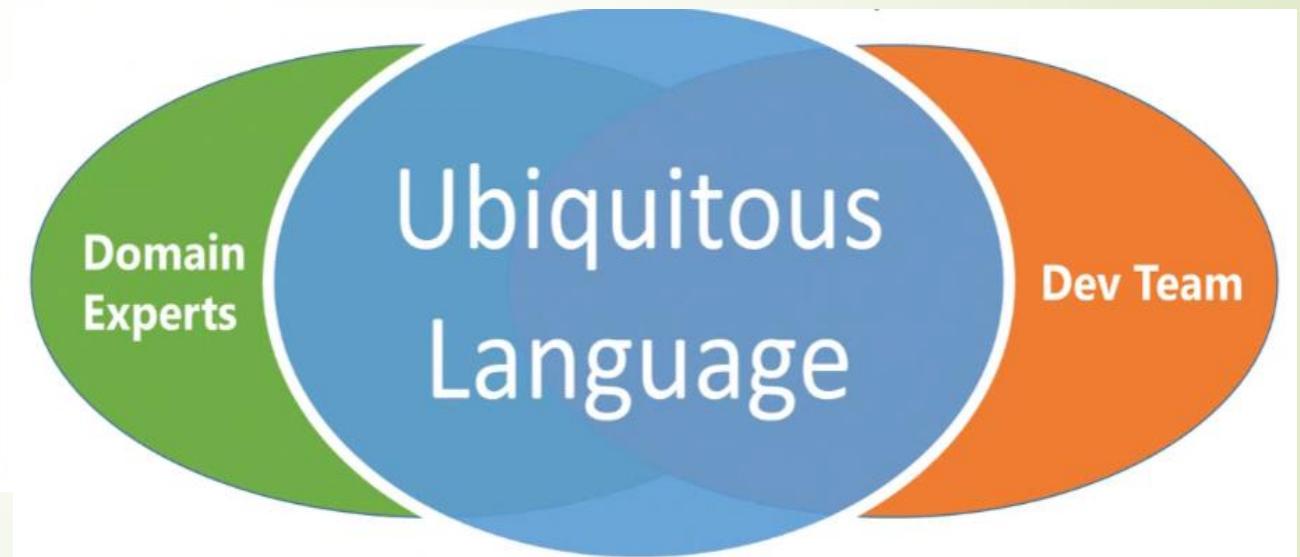
Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

Atenção!!!
Deixar de extrair a
linguagem ubíqua é
um dos piores erros.

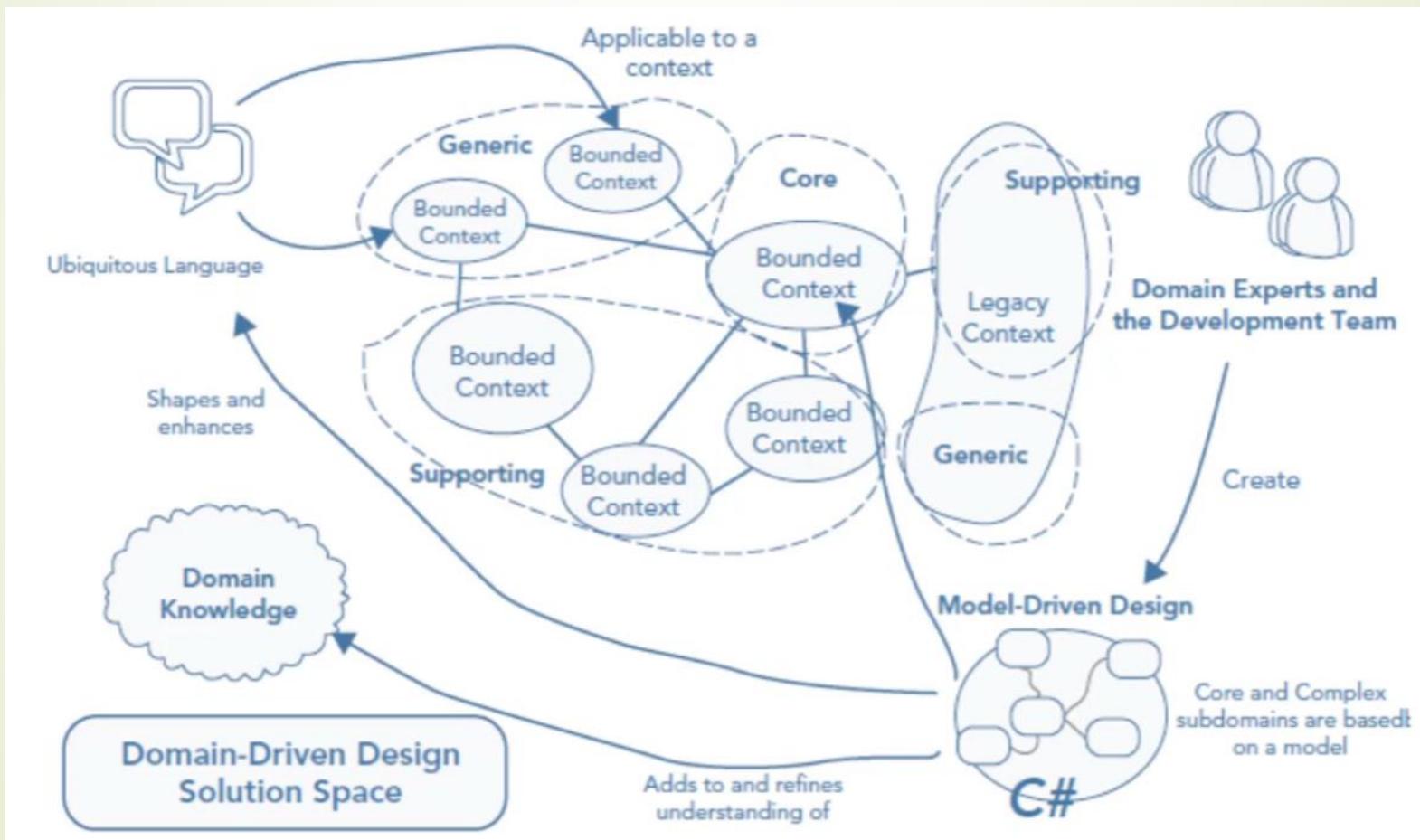


Primeiro passo extrair a linguagem ubíqua



Projeto de Bloco: Engenharia de Software e Modelagem

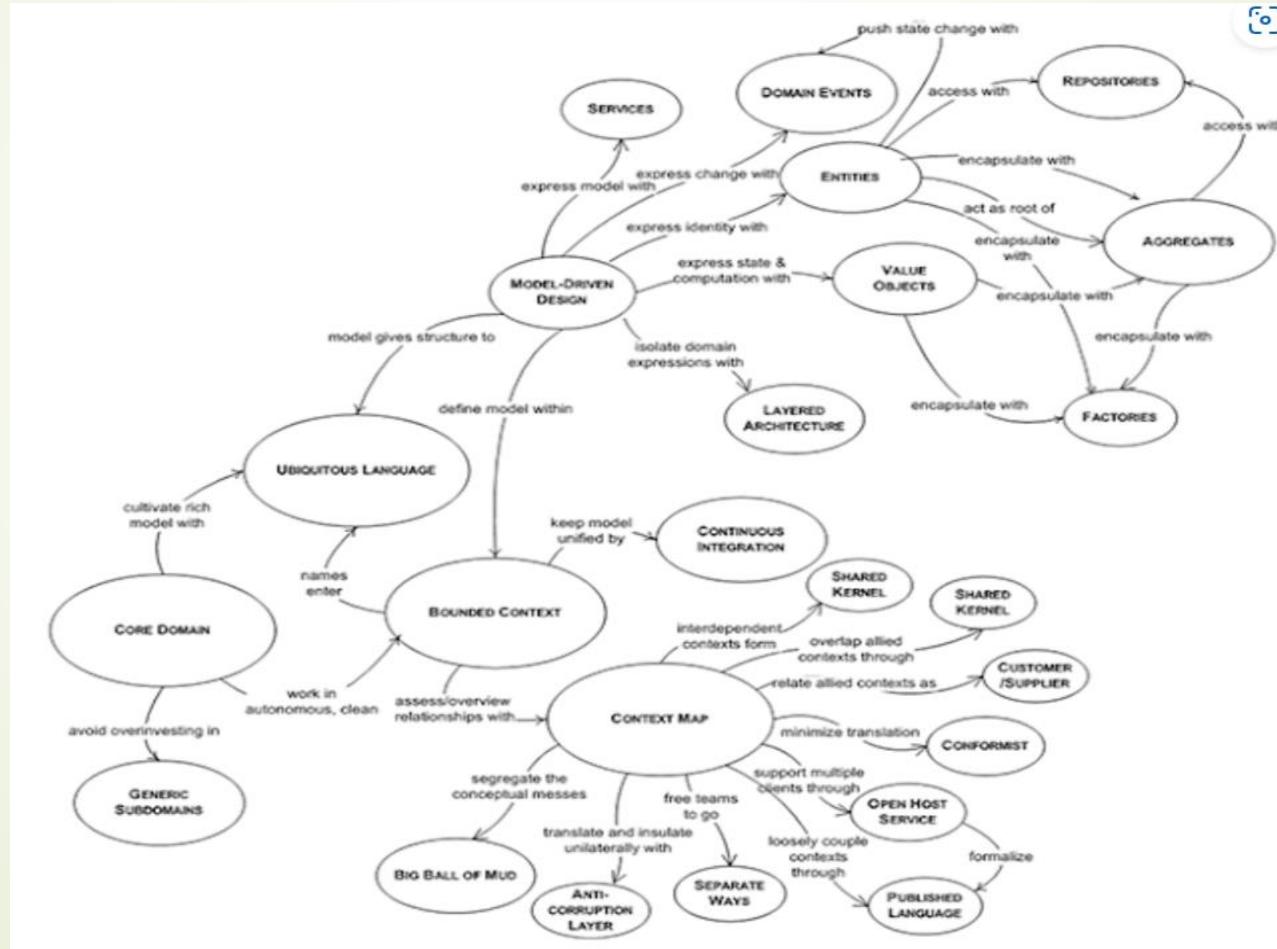
Aplicar os conceitos de DDD



Projeto de Bloco: Engenharia de Software e Modelagem

Mapa de Aprendizado

Aplicar os conceitos de DDD



Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD



- A loja virtual exibirá um catálogo de produtos de diversas categorias.
- Um cliente pode realizar um pedido contendo 1 ou N produtos.
- A loja realizará as vendas através de pagamento por cartão de crédito.
- O cliente irá realizar o seu cadastro para poder fazer pedidos.
- O cliente irá confirmar o pedido, endereço de entrega, escolher o tipo de frete e realizar o pagamento.
- Após o pagamento o pedido mudará de status conforme resposta da transação via cartão.
- Ocorrerá a emissão da nota fiscal logo após a confirmação de pagamento do pedido.

Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD



- A loja virtual exibirá um **catálogo** de **produtos** de diversas **categorias**.
- Um **cliente** pode realizar um **pedido** contendo 1 ou N produtos.
- A loja realizará as **vendas** através de **pagamento** por **cartão de crédito**.
- O cliente irá realizar o seu **cadastro** para poder fazer pedidos.
- O cliente irá confirmar o pedido, **endereço** de entrega, escolher o tipo de **frete** e realizar o pagamento.
- Após o pagamento o pedido mudará de **status** conforme resposta da **transação** via cartão.
- Ocorrerá a emissão da **nota fiscal** logo após a confirmação de pagamento do pedido.

Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

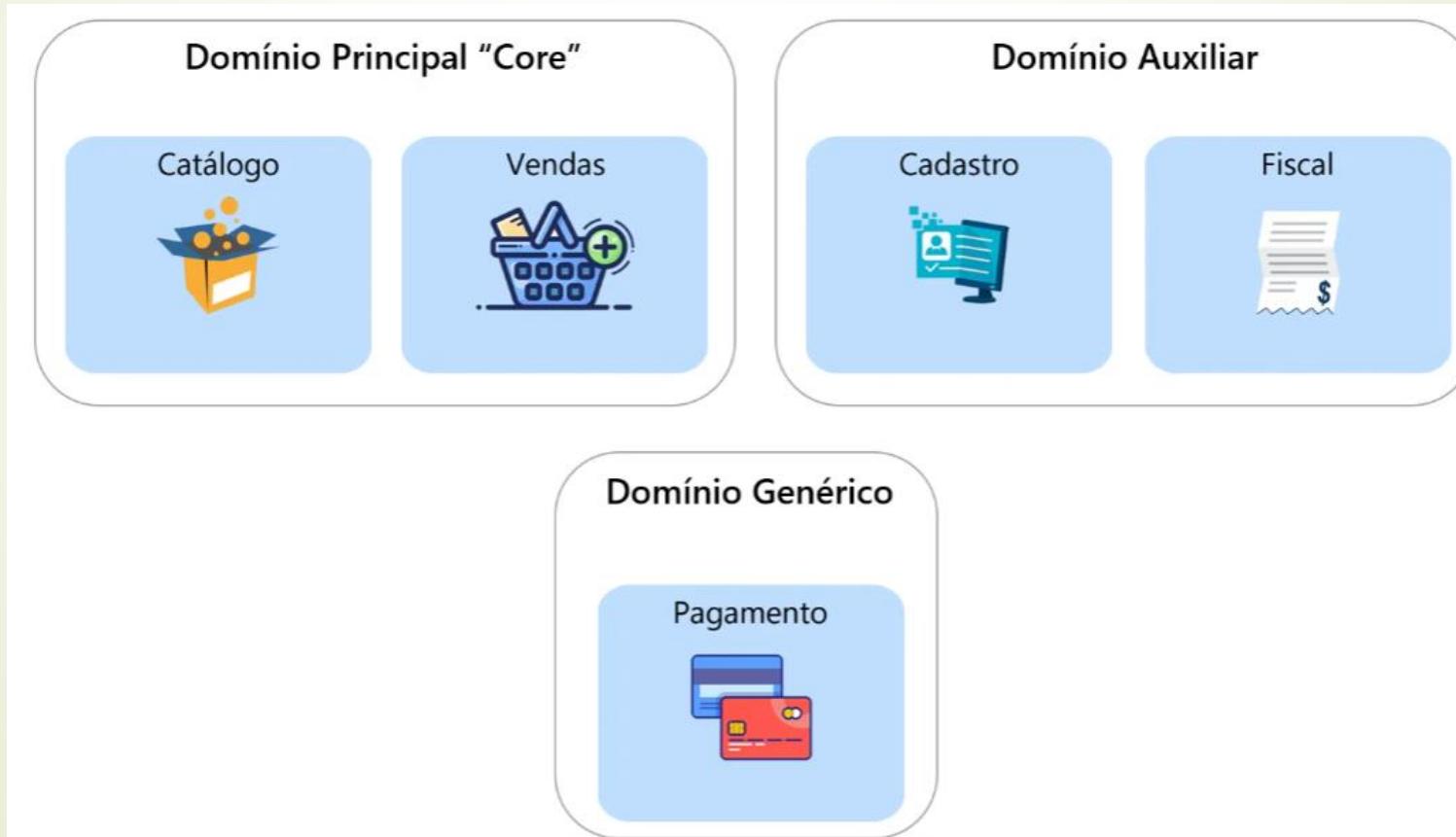


- A loja virtual exibirá um catálogo de produtos de diversas categorias.
- Um cliente pode realizar um pedido contendo 1 ou N produtos.
- A loja realizará as vendas através de pagamento por cartão de crédito.
- O cliente irá realizar o seu cadastro para poder fazer pedidos.
- O cliente irá confirmar o pedido endereço de entrega, escolher o tipo de frete e realizar o pagamento.
- Após o pagamento o pedido mudará de status conforme resposta da transação via cartão.
- Ocorrerá a emissão da nota fiscal logo após a confirmação de pagamento do pedido.

Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

Tipos de Domínios



Projeto de Bloco: Engenharia de Software e Modelagem

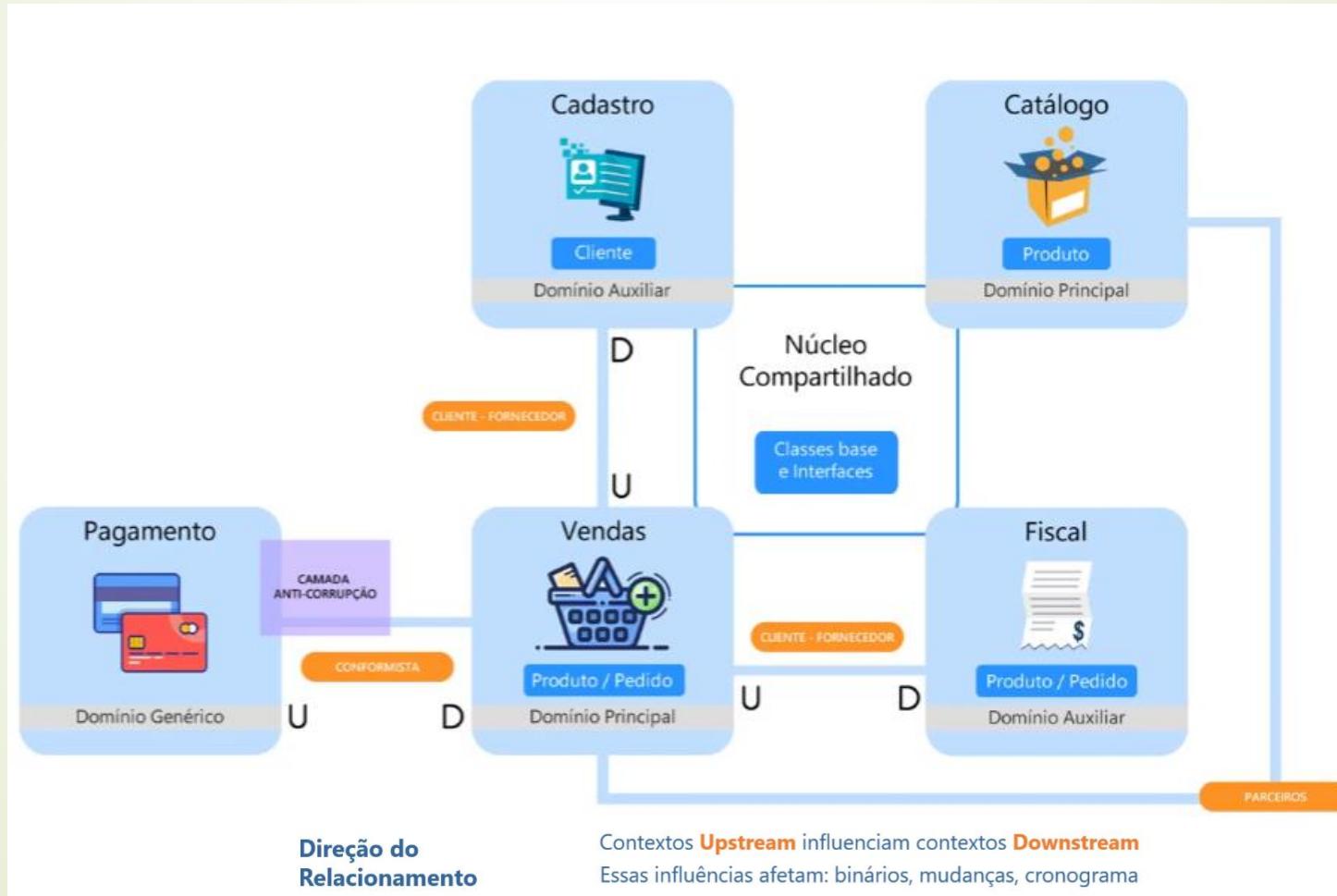
Aplicar os conceitos de DDD



Projeto de Bloco: Engenharia de Software e Modelagem

Mapa de Contextos

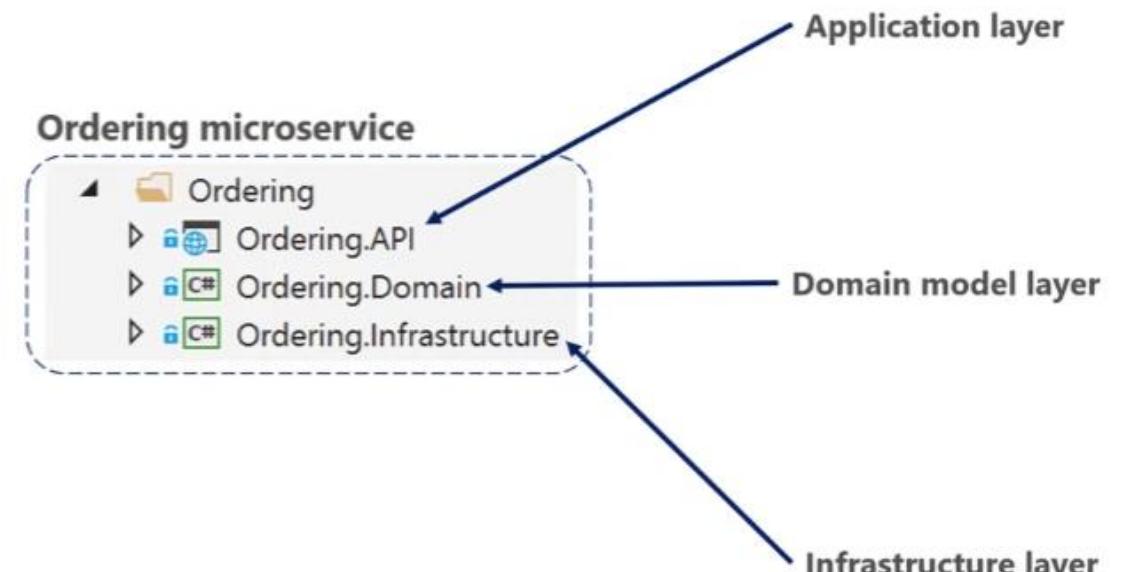
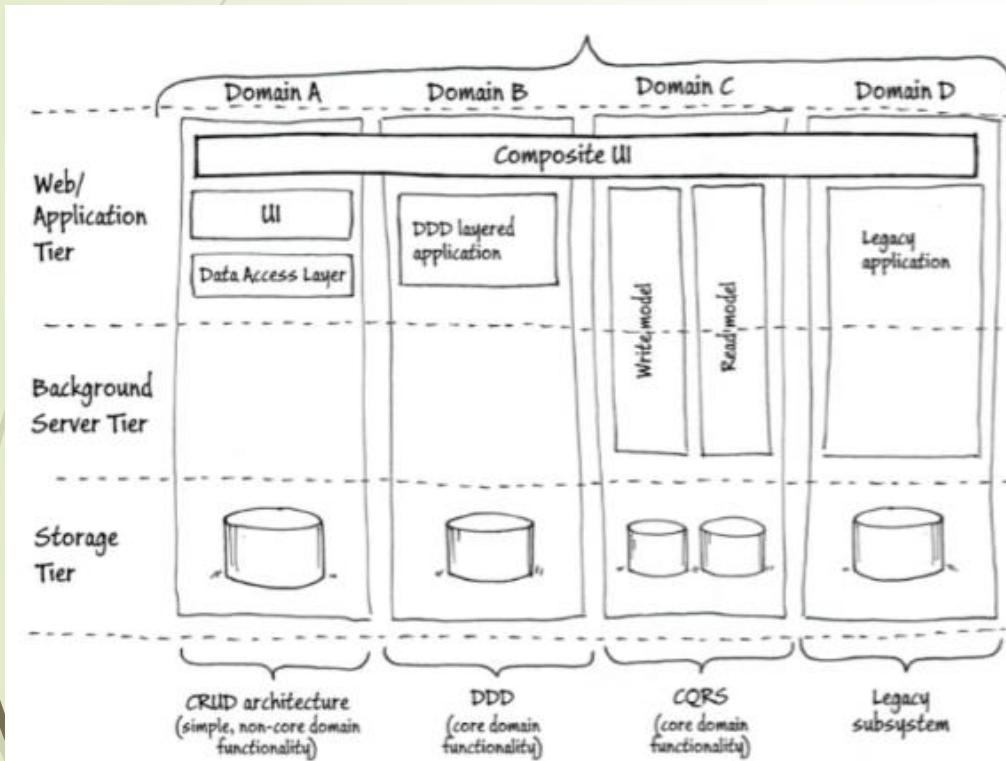
Aplicar os conceitos de DDD



Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

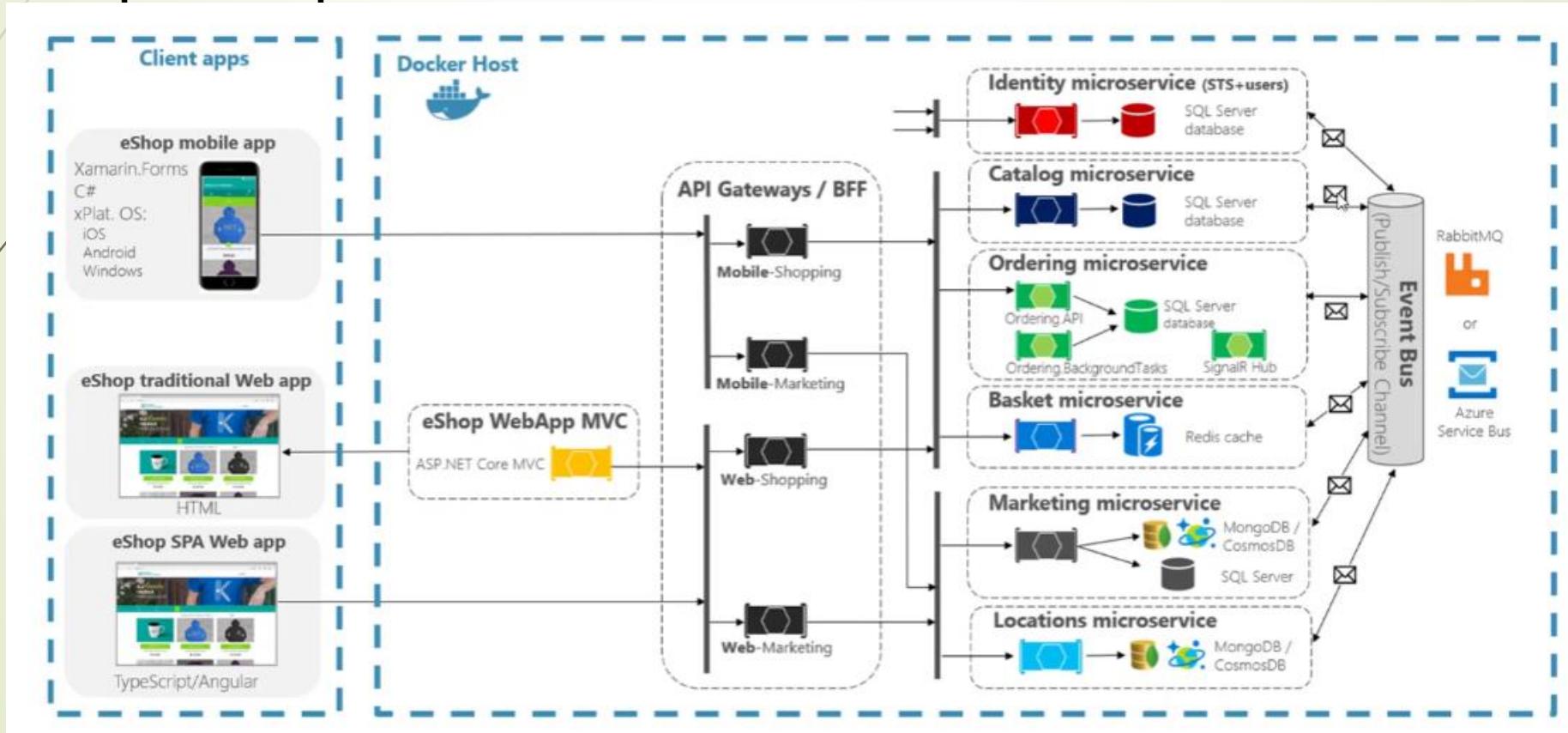
Definir arquitetura dos contextos



Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

Exemplo de arquitetura dos contextos





Projeto de Bloco: Engenharia de Software e Modelagem

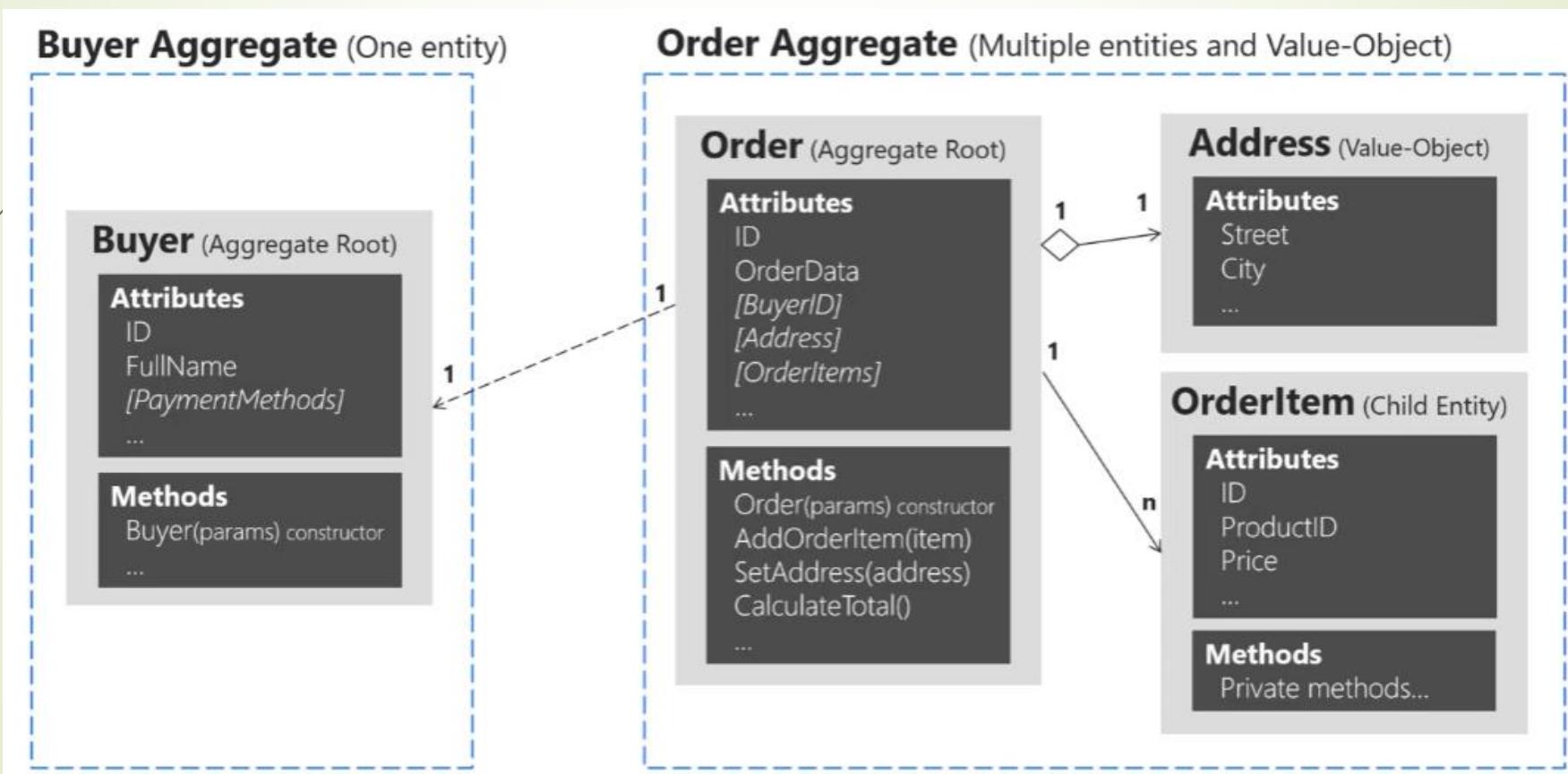


MODELAGEM TÁTICA

Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

Agregação



Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

Interface de marcação

```
1 namespace NerdStore.Core.DomainObjects
2 {
3     | 4 referências
4         public interface IAggregateRoot { }
```



```
4 namespace NerdStore.Core.Data
5 {
6     | 3 referências
7         public interface IRepository<T> : IDisposable where T : IAggregateRoot
8             {
9                 | 18 referências
10                    IUnitOfWork UnitOfWork { get; }
11                }
```

Raiz de agregação

```
7  namespace NerdStore.Vendas.Domain
8  {
9      21 referências
10     public class Pedido : Entity, IAggregateRoot
11     {
12         3 referências
13         public int Código { get; private set; }
14         8 referências
15         public Guid ClienteId { get; private set; }
16         3 referências
17         public Guid? VoucherId { get; private set; }
18         3 referências
19         public bool VoucherUtilizado { get; private set; }
20         4 referências
21         public decimal Desconto { get; private set; }
22         8 referências
23         public decimal ValorTotal { get; private set; }
24         1 referência
25         public DateTime DataCadastro { get; private set; }
26         8 referências
27         public PedidoStatus PedidoStatus { get; private set; }

28         private readonly List<PedidoItem> _pedidoItems;
29         10 referências
30         public IReadOnlyCollection<PedidoItem> PedidoItems => _pedidoItems;

31         // EF Rel.
32         9 referências
33         public Voucher Voucher { get; private set; }
34     }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

Objeto de valor

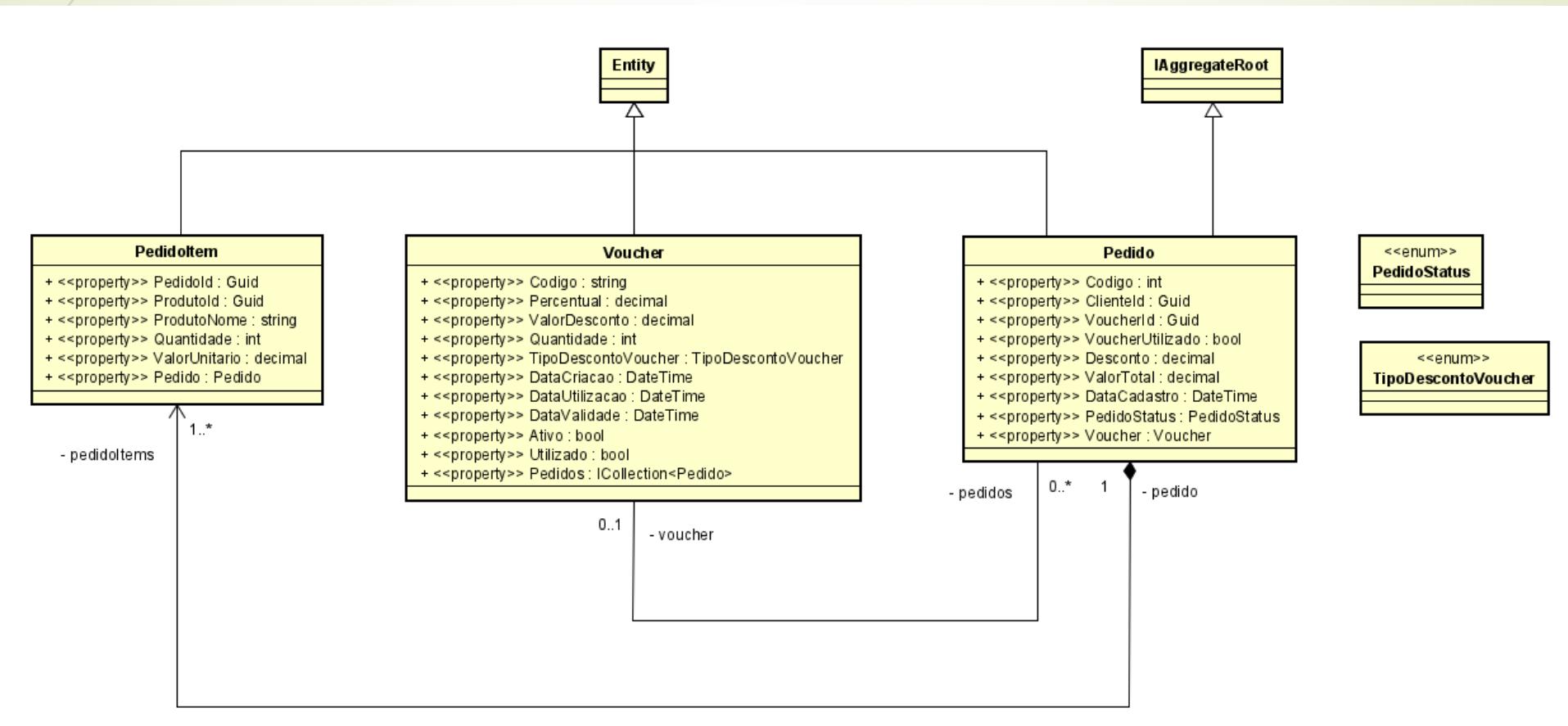
```
1 using System;
2 using NerdStore.Core.DomainObjects;
3
4 namespace NerdStore.Catalogo.Domain
5 {
6     public class Produto : Entity, IAggregateRoot
7     {
8         public Guid CategoriaId { get; private set; }
9         public string Nome { get; private set; }
10        public string Descricao { get; private set; }
11        public bool Ativo { get; private set; }
12        public decimal Valor { get; private set; }
13        public DateTime DataCadastro { get; private set; }
14        public string Imagem { get; private set; }
15        public int QuantidadeEstoque { get; private set; }
16        public Dimensoes Dimensoes { get; private set; }
17        public Categoria Categoria { get; private set; }
18    }
```

```
1 using NerdStore.Core.DomainObjects;
2
3 namespace NerdStore.Catalogo.Domain
4 {
5     public class Dimensoes
6     {
7         public decimal Altura { get; private set; }
8         public decimal Largura { get; private set; }
9         public decimal Profundidade { get; private set; }
10    }
11    public Dimensoes(decimal altura, decimal largura, decimal profundidade)
12    {
13        Validacoes.ValidarSeMenorQue(valor: altura, minimo: 1, mensagem: "O campo Altura não pode ser menor ou igual a 0");
14        Validacoes.ValidarSeMenorQue(valor: largura, minimo: 1, mensagem: "O campo Largura não pode ser menor ou igual a 0");
15        Validacoes.ValidarSeMenorQue(valor: profundidade, minimo: 1, mensagem: "O campo Profundidade não pode ser menor ou igual a 0");
16
17        Altura = altura;
18        Largura = largura;
19        Profundidade = profundidade;
20    }
21
22    public string DescricaoFormatada()
23    {
24        return $"{LxAxP: {Largura} x {Altura} x {Profundidade}}";
25    }
26
27    public override string ToString()
28    {
29        return DescricaoFormatada();
30    }
31}
32}
```

Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

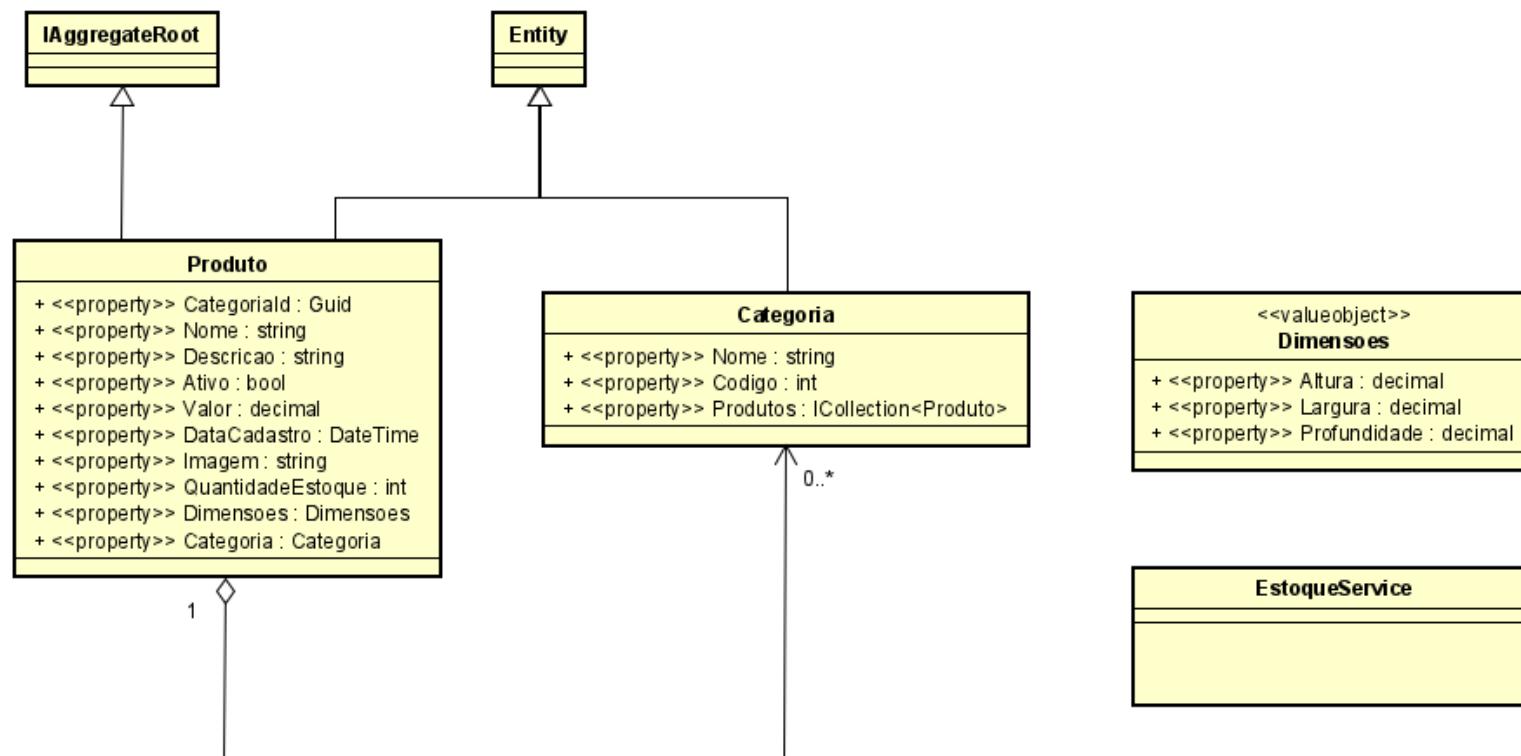
Modelo Conceitual – Contexto Vendas



Projeto de Bloco: Engenharia de Software e Modelagem

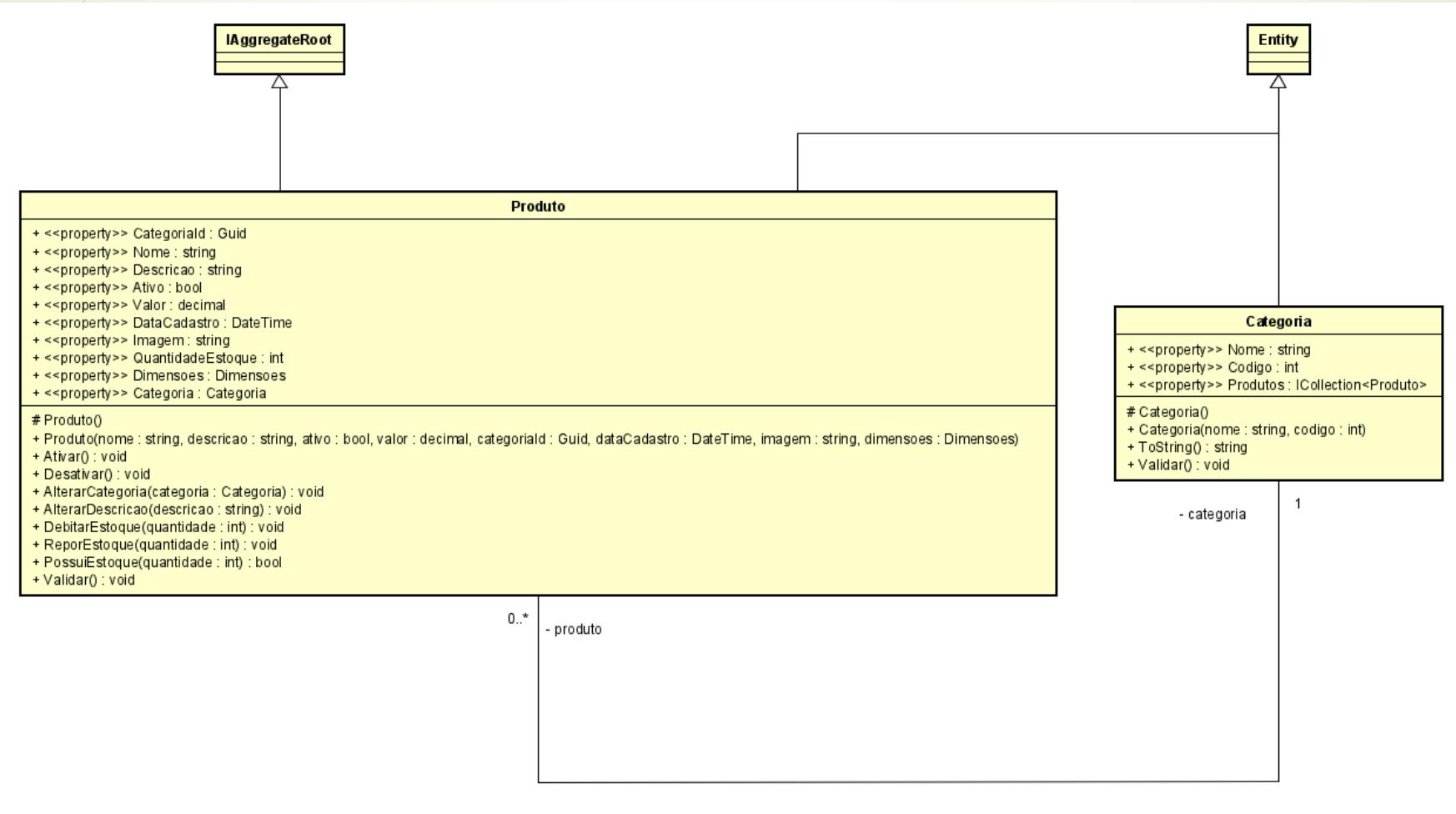
Aplicar os conceitos de DDD

Modelo Conceitual – Contexto Catálogo



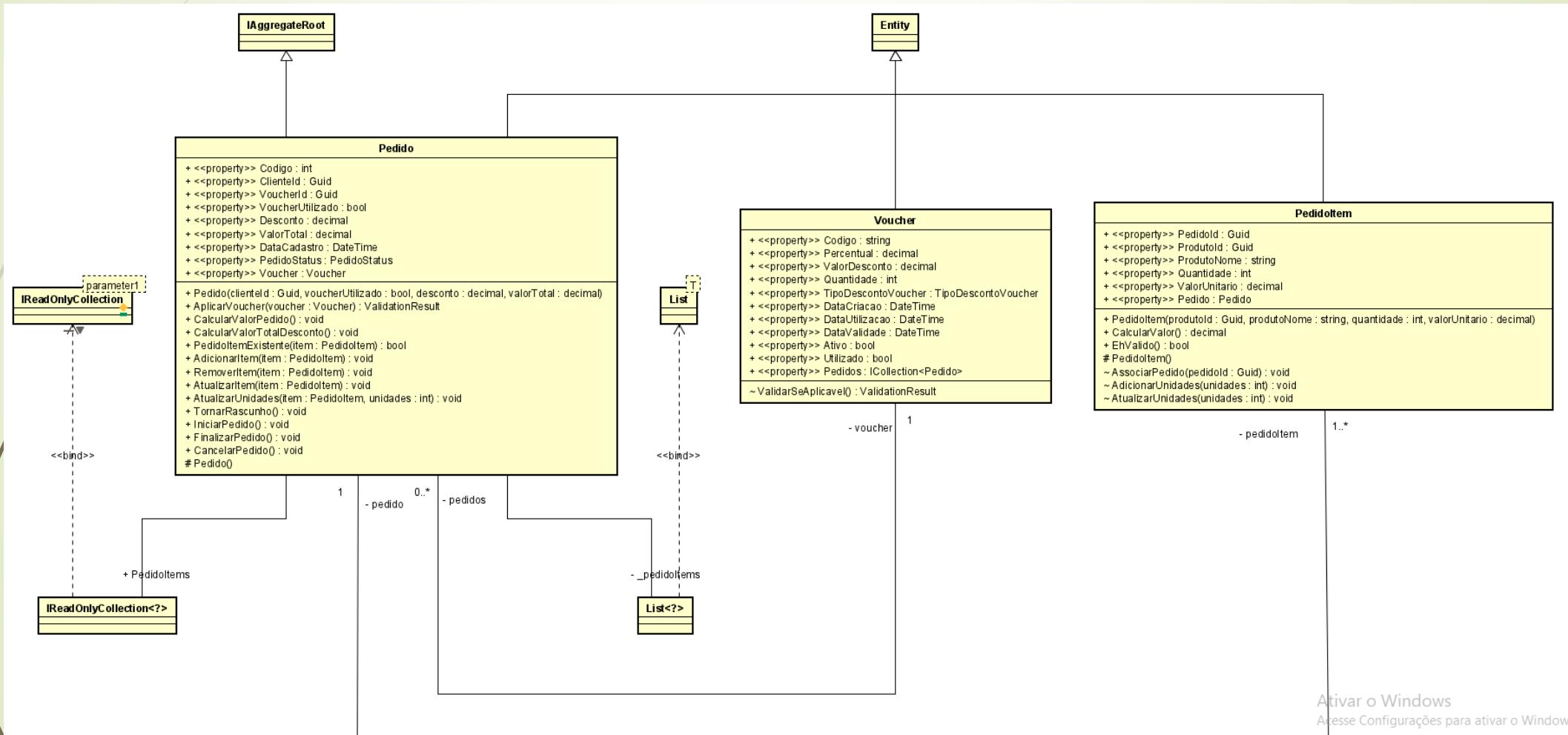
Projeto de Bloco: Engenharia de Software e Modelagem

Diagrama de Classe – Contexto Catálogo



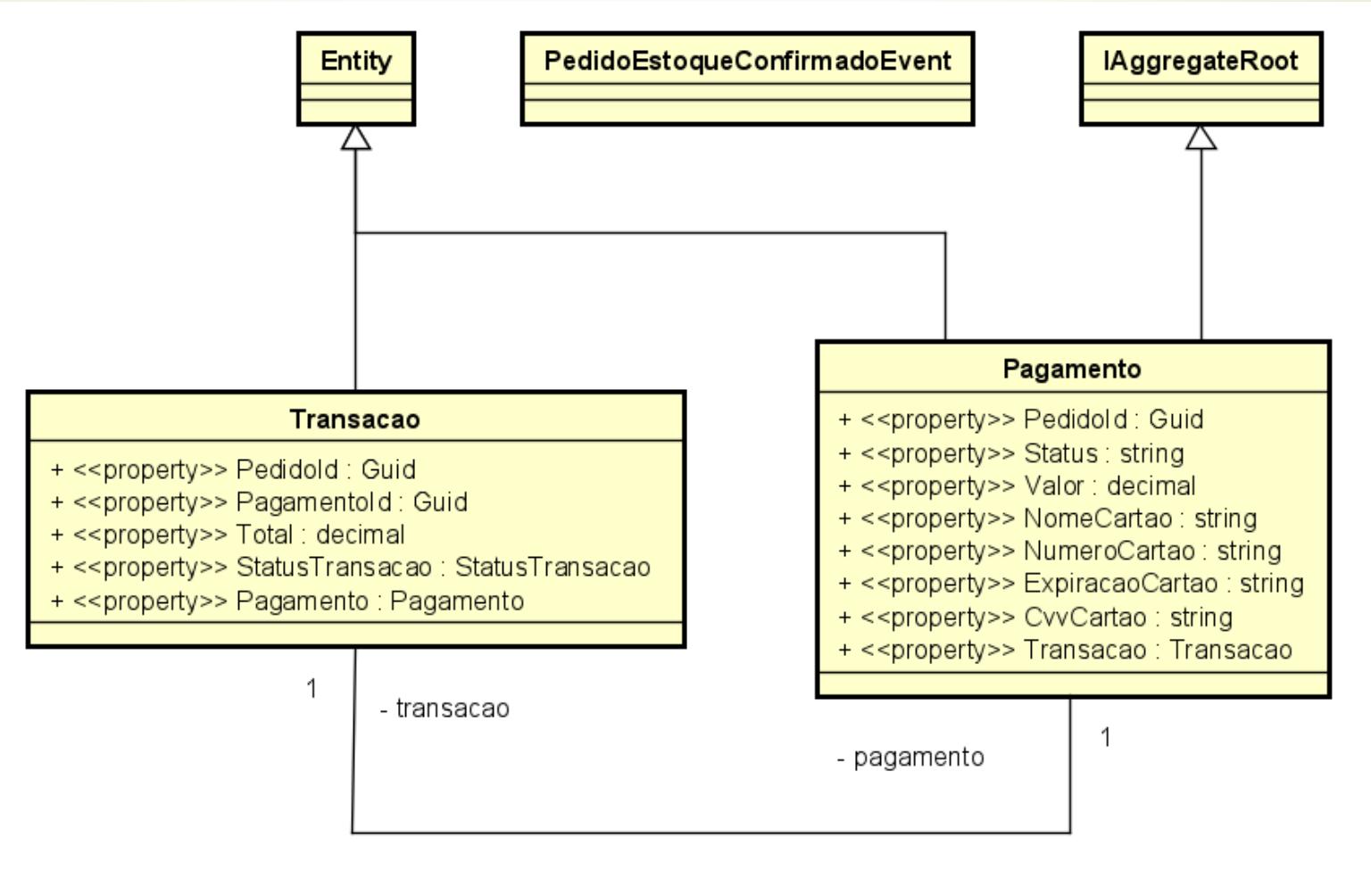
Projeto de Bloco: Engenharia de Software e Modelagem

Diagrama de Classe – Contexto Vendas



Projeto de Bloco: Engenharia de Software e Modelagem

Diagrama de Classe – Contexto Pagamentos



Projeto de Bloco: Engenharia de Software e Modelagem

Rel. OneToMany – Contexto Catálogo

```
1  using System;
2  using NerdStore.Core.DomainObjects;
3
4  namespace NerdStore.Catalogo.Domain
5  {
6      - referências
7      public class Produto : Entity, IAggregateRoot
8      {
9          - referências
10         public Guid CategoriaId { get; private set; }
11
12         - referências
13         public string Nome { get; private set; }
14
15         - referências
16         public string Descricao { get; private set; }
17
18         - referências
19         public bool Ativo { get; private set; }
20
21         - referências
22         public decimal Valor { get; private set; }
23
24         - referências
25         public DateTime DataCadastro { get; private set; }
26
27         - referências
28         public string Imagem { get; private set; }
29
30         - referências
31         public int QuantidadeEstoque { get; private set; }
32
33         - referências
34         public Dimensoes Dimensoes { get; private set; }
35
36         - referências
37         public Categoria Categoria { get; private set; }
38
39         - referências
40         protected Produto() { }
41
42         - referências
43         public Produto(string nome,
44                         string descricao,
45                         bool ativo,
46                         decimal valor,
47                         Guid categoriaId,
48                         DateTime dataCadastro,
49                         string imagem,
50                         Dimensoes dimensoes)
51
52             {
53                 CategoriaId = categoriaId;
54                 Nome = nome;
55                 Descricao = descricao;
56                 Ativo = ativo;
57                 Valor = valor;
58                 DataCadastro = dataCadastro;
59                 Imagem = imagem;
60                 Dimensoes = dimensoes;
61
62                 Validar();
63             }
64
65         - referências
66     }
67 }
```

```
1  using System.Collections.Generic;
2  using NerdStore.Core.DomainObjects;
3
4  namespace NerdStore.Catalogo.Domain
5  {
6      - referências
7      public class Categoria : Entity
8      {
9          - referências
10         public string Nome { get; private set; }
11
12         - referências
13         public int Código { get; private set; }
14
15         // EF Relation
16         - referências
17         public ICollection<Produto> Produtos { get; set; }
18
19         - referências
20         protected Categoria() { }
21
22         - referências
23         public Categoria(string nome, int código)
24         {
25             Nome = nome;
26             Código = código;
27
28             Validar();
29         }
30
31         - referências
32         public override string ToString()
33         {
34             return $"{Nome} - {Código}";
35         }
36
37         - referências
38         public void Validar()
39         {
40             Validações.ValidarSeVazio(Nome, "O campo Nome da categoria não pode estar vazio");
41             Validações.ValidarSeIgual(Código, 0, "O campo Código não pode ser 0");
42         }
43     }
44 }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Rel. OneToMany – Contexto Vendas

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using FluentValidation.Results;
5  using NerdStore.Core.DomainObjects;
6
7  namespace NerdStore.Vendas.Domain
8  {
9      - referências
10     public class Pedido : Entity, IAggregateRoot
11     {
12         - referências
13         public int Codigo { get; private set; }
14         - referências
15         public Guid ClienteId { get; private set; }
16         - referências
17         public Guid? VoucherId { get; private set; }
18         - referências
19         public bool VoucherUtilizado { get; private set; }
20         - referências
21         public decimal Desconto { get; private set; }
22         - referências
23         public decimal ValorTotal { get; private set; }
24         - referências
25         public DateTime DataCadastro { get; private set; }
26         - referências
27         public PedidoStatus PedidoStatus { get; private set; }
28
29         private readonly List<PedidoItem> _pedidoItems;
30         - referências
31         public IReadOnlyCollection<PedidoItem> PedidoItems => _pedidoItems;
32
33         // EF Rel.
34         - referências
35         public Voucher Voucher { get; private set; }
36
37         - referências
38         public Pedido(Guid clienteId,
39                     bool voucherUtilizado,
40                     decimal desconto,
41                     decimal valorTotal)
42         {
43             ClienteId = clienteId;
44             VoucherUtilizado = voucherUtilizado;
45             Desconto = desconto;
46             ValorTotal = valorTotal;
47             _pedidoItems = new List<PedidoItem>();
48         }
49
50         - referências
51         protected Pedido()
52         {
53             _pedidoItems = new List<PedidoItem>();
54         }
55     }
56 }
```

```
1  using System;
2  using NerdStore.Core.DomainObjects;
3
4  namespace NerdStore.Vendas.Domain
5  {
6      - referências
7      public class PedidoItem : Entity
8      {
9          - referências
10         public Guid PedidoId { get; private set; }
11         - referências
12         public Guid ProdutoId { get; private set; }
13         - referências
14         public string ProdutoNome { get; private set; }
15         - referências
16         public int Quantidade { get; private set; }
17         - referências
18         public decimal ValorUnitario { get; private set; }
19
20         // EF Rel.
21         - referências
22         public Pedido Pedido { get; set; }
23
24         - referências
25         public PedidoItem(Guid produtoId,
26                           string produtoNome,
27                           int quantidade,
28                           decimal valorUnitario)
29         {
30             ProdutoId = produtoId;
31             ProdutoNome = produtoNome;
32             Quantidade = quantidade;
33             ValorUnitario = valorUnitario;
34         }
35
36         - referências
37         protected PedidoItem() { }
38
39         - referências
40         internal void AssociarPedido(Guid pedidoId)
41         {
42             PedidoId = pedidoId;
43         }
44
45         - referências
46         public decimal CalcularValor()
47         {
48             return Quantidade * ValorUnitario;
49         }
50     }
51 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using FluentValidation;
4  using FluentValidation.Results;
5  using NerdStore.Core.DomainObjects;
6
7  namespace NerdStore.Vendas.Domain
8  {
9      - referências
10     public class Voucher : Entity
11     {
12         - referências
13         public string Codigo { get; private set; }
14         - referências
15         public decimal? Percentual { get; private set; }
16         - referências
17         public decimal? ValorDesconto { get; private set; }
18         - referências
19         public int Quantidade { get; private set; }
20         - referências
21         public TipoDescontoVoucher TipoDescontoVoucher { get; private set; }
22         - referências
23         public DateTime DataCriacao { get; private set; }
24         - referências
25         public DateTime? DataUtilizacao { get; private set; }
26         - referências
27         public bool Ativo { get; private set; }
28         - referências
29         public bool Utilizado { get; private set; }
30
31         // EF Rel.
32         - referências
33         public ICollection<Pedido> Pedidos { get; set; }
34
35         - referências
36         internal ValidationResult ValidarSeAplicavel()
37         {
38             return new VoucherAplicavelValidation().Validate(this);
39         }
40     }
41 }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Rel. OneToOne – Contexto Pagamentos

```
1  using System;
2  using NerdStore.Core.DomainObjects;
3
4  namespace NerdStore.Pagamentos.Business
5  {
6      public class Pagamento : Entity, IAggregateRoot
7      {
8          public Guid PedidoId { get; set; }
9          public string Status { get; set; }
10         public decimal Valor { get; set; }
11
12         public string NomeCartao { get; set; }
13         public string NumeroCartao { get; set; }
14         public string ExpiracaoCartao { get; set; }
15         public string CvvCartao { get; set; }
16
17         // EF. Rel.
18         public Transacao Transacao { get; set; }
19     }
20 }
21
```

```
1  using System;
2  using NerdStore.Core.DomainObjects;
3
4  namespace NerdStore.Pagamentos.Business
5  {
6      public class Transacao : Entity
7      {
8          public Guid PedidoId { get; set; }
9          public Guid PagamentoId { get; set; }
10         public decimal Total { get; set; }
11         public StatusTransacao StatusTransacao { get; set; }
12
13         // EF. Rel.
14         public Pagamento Pagamento { get; set; }
15     }
16 }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Relacionamento Extends

Projeto de Bloco: Engenharia de Software e Modelagem

Herança – Contexto Catálogo

```
1  using System;
2  using NerdStore.Core.DomainObjects;
3
4  namespace NerdStore.Catalogo.Domain
5  {
6      public class Produto : Entity, IAggregateRoot
7      {
8          public Guid CategoriaId { get; private set; }
9          public string Nome { get; private set; }
10         public string Descricao { get; private set; }
11         public bool Ativo { get; private set; }
12         public decimal Valor { get; private set; }
13         public DateTime DataCadastro { get; private set; }
14         public string Imagem { get; private set; }
15         public int QuantidadeEstoque { get; private set; }
16         public Dimensoes Dimensoes { get; private set; }
17         public Categoria Categoria { get; private set; }
18
19     }
```

```
1  using System;
2  using System.Collections.Generic;
3  using NerdStore.Core.Messages;
4
5  namespace NerdStore.Core.DomainObjects
6  {
7      public abstract class Entity
8      {
9          public Guid Id { get; set; }
10         private List<Event> _notificacoes;
11         public IReadOnlyCollection<Event> Notificacoes => _notificacoes?.AsReadOnly();
12
13         protected Entity()
14         {
15             Id = Guid.NewGuid();
16         }
17
18         public void AdicionarEvento(Event evento)
19         {
20             _notificacoes = _notificacoes ?? new List<Event>();
21             _notificacoes.Add(evento);
22         }
23
24         public void RemoverEvento(Event eventItem)
25         {
26             _notificacoes?.Remove(eventItem);
27         }
28
29         public void LimparEventos()
30         {
31             _notificacoes?.Clear();
32         }
33
34         public override bool Equals(object obj)
35         {
36             var compareTo = obj as Entity;
37
38             if (ReferenceEquals(this, compareTo)) return true;
39             if (ReferenceEquals(null, compareTo)) return false;
40
41             return Id.Equals(compareTo.Id);
42         }
43     }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Encapsulamento – Contexto Vendas

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using FluentValidation.Results;
5  using NerdStore.Core.DomainObjects;
6
7  namespace NerdStore.Vendas.Domain
8  {
9      - referências
10     public class Pedido : Entity, IAggregateRoot
11     {
12         - referências
13         public int Código { get; private set; }
14         - referências
15         public Guid ClienteId { get; private set; }
16         - referências
17         public Guid? VoucherId { get; private set; }
18         - referências
19         public bool VoucherUtilizado { get; private set; }
20         - referências
21         public decimal Desconto { get; private set; }
22         - referências
23         public decimal ValorTotal { get; private set; }
24         - referências
25         public DateTime DataCadastro { get; private set; }
26         - referências
27         public PedidoStatus PedidoStatus { get; private set; }
28
29         private readonly List<PedidoItem> _pedidoItems;
30
31         public Pedido(Guid clienteId,
32             bool voucherUtilizado,
33             decimal desconto,
34             decimal valorTotal)
35         {
36             ClienteId = clienteId;
37             VoucherUtilizado = voucherUtilizado;
38             Desconto = desconto;
39             ValorTotal = valorTotal;
40             _pedidoItems = new List<PedidoItem>();
41         }
42
43         - referências
44         public ValidationResult AplicarVoucher(Voucher voucher)
45         {
46             var validationResult = voucher.ValidarSeAplicavel();
47             if (!validationResult.IsValid) return validationResult;
48
49             Voucher = voucher;
50             VoucherUtilizado = true;
51             CalcularValorPedido();
52
53             return validationResult;
54         }
55
56         - referências
57         public void CalcularValorPedido()
58         {
59             ValorTotal = PedidoItems.Sum(p => p.CalcularValor());
60             CalcularValorTotalDesconto();
61
62         }
63
64         - referências
65         public void CalcularValorTotalDesconto()
66         {
67             if (!VoucherUtilizado) return;
68
69             decimal desconto = 0;
70             var valor = ValorTotal;
71
72             if (Voucher.TipoDescontoVoucher == TipoDescontoVoucher.Porcentagem)
73             {
74                 if (Voucher.Percentual.HasValue)
75                 {
76                     desconto = (valor * Voucher.Percentual.Value) / 100;
77                     valor -= desconto;
78                 }
79                 else
80                 {
81                     if (Voucher.ValorDesconto.HasValue)
82                     {
83                         desconto = Voucher.ValorDesconto.Value;
84                         valor -= desconto;
85                     }
86                 }
87             }
88
89             ValorTotal = valor < 0 ? 0 : valor;
90             Desconto = desconto;
91         }
92
93         - referências
94         public PedidoItem Existente(PedidoItem item)
95         {
96             return _pedidoItems.Any(p => p.ProdutoId == item.ProdutoId);
97         }
98
99         - referências
100        public void AdicionarItem(PedidoItem item)
101        {
102            if (!item.EHValido()) return;
103            item.AssociarPedido(Id);
104
105            if (PedidoItemExistente(item))
106            {
107                var itemExistente = _pedidoItems.FirstOrDefault(p => p.ProdutoId == item.ProdutoId);
108                itemExistente.AdicionarUnidades(item.Unidades);
109                item = itemExistente;
110
111                _pedidoItems.Remove(itemExistente);
112            }
113
114            item.CalcularValor();
115            _pedidoItems.Add(item);
116
117            CalcularValorPedido();
118
119            - referências
120            public void RemoverItem(PedidoItem item)
121            {
122                if (!item.EHValido()) return;
123
124                var itemExistente = _pedidoItems.FirstOrDefault(p => p.ProdutoId == item.ProdutoId);
125
126                if (itemExistente == null) throw new DomainException("O item não pertence ao pedido");
127                _pedidoItems.Remove(itemExistente);
128
129                CalcularValorPedido();
130            }
131
132            - referências
133        }
134    }
135}
```

```
43         - referências
44         public ValidationResult AplicarVoucher(Voucher voucher)
45         {
46             var validationResult = voucher.ValidarSeAplicavel();
47             if (!validationResult.IsValid) return validationResult;
48
49             Voucher = voucher;
50             VoucherUtilizado = true;
51             CalcularValorPedido();
52
53             return validationResult;
54         }
55
56         - referências
57         public void CalcularValorPedido()
58         {
59             ValorTotal = PedidoItems.Sum(p => p.CalcularValor());
60             CalcularValorTotalDesconto();
61
62         }
63
64         - referências
65         public void CalcularValorTotalDesconto()
66         {
67             if (!VoucherUtilizado) return;
68
69             decimal desconto = 0;
70             var valor = ValorTotal;
71
72             if (Voucher.TipoDescontoVoucher == TipoDescontoVoucher.Porcentagem)
73             {
74                 if (Voucher.Percentual.HasValue)
75                 {
76                     desconto = (valor * Voucher.Percentual.Value) / 100;
77                     valor -= desconto;
78                 }
79                 else
80                 {
81                     if (Voucher.ValorDesconto.HasValue)
82                     {
83                         desconto = Voucher.ValorDesconto.Value;
84                         valor -= desconto;
85                     }
86                 }
87             }
88
89             ValorTotal = valor < 0 ? 0 : valor;
90             Desconto = desconto;
91         }
92
93         - referências
94         public PedidoItem Existente(PedidoItem item)
95         {
96             return _pedidoItems.Any(p => p.ProdutoId == item.ProdutoId);
97         }
98
99         - referências
100        public void AdicionarItem(PedidoItem item)
101        {
102            if (!item.EHValido()) return;
103            item.AssociarPedido(Id);
104
105            if (PedidoItemExistente(item))
106            {
107                var itemExistente = _pedidoItems.FirstOrDefault(p => p.ProdutoId == item.ProdutoId);
108                itemExistente.AdicionarUnidades(item.Unidades);
109                item = itemExistente;
110
111                _pedidoItems.Remove(itemExistente);
112            }
113
114            item.CalcularValor();
115            _pedidoItems.Add(item);
116
117            CalcularValorPedido();
118
119            - referências
120            public void RemoverItem(PedidoItem item)
121            {
122                if (!item.EHValido()) return;
123
124                var itemExistente = _pedidoItems.FirstOrDefault(p => p.ProdutoId == item.ProdutoId);
125
126                if (itemExistente == null) throw new DomainException("O item não pertence ao pedido");
127                _pedidoItems.Remove(itemExistente);
128
129                CalcularValorPedido();
130            }
131
132            - referências
133        }
134    }
135}
```

```
89         - referências
90         public PedidoItem Existente(PedidoItem item)
91         {
92             return _pedidoItems.Any(p => p.ProdutoId == item.ProdutoId);
93         }
94
95         - referências
96         public void AdicionarItem(PedidoItem item)
97         {
98             if (!item.EHValido()) return;
99             item.AssociarPedido(Id);
100
101             if (PedidoItemExistente(item))
102             {
103                 var itemExistente = _pedidoItems.FirstOrDefault(p => p.ProdutoId == item.ProdutoId);
104                 itemExistente.AdicionarUnidades(item.Unidades);
105                 item = itemExistente;
106
107                 _pedidoItems.Remove(itemExistente);
108             }
109
110             item.CalcularValor();
111             _pedidoItems.Add(item);
112
113             CalcularValorPedido();
114
115             - referências
116             public void RemoverItem(PedidoItem item)
117             {
118                 if (!item.EHValido()) return;
119
120                 var itemExistente = _pedidoItems.FirstOrDefault(p => p.ProdutoId == item.ProdutoId);
121
122                 if (itemExistente == null) throw new DomainException("O item não pertence ao pedido");
123                 _pedidoItems.Remove(itemExistente);
124
125                 CalcularValorPedido();
126             }
127
128             - referências
129         }
130     }
131
132     - referências
133 }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Polimorfismo – Contexto Catálogo e Vendas

Projeto de Bloco: Engenharia de Software e Modelagem

Interfaces – Contexto Catálogo e Vendas

```
1  using System;
2  using System.Collections.Generic;
3  using System.Threading.Tasks;
4  using NerdStore.Core.Data;
5
6  namespace NerdStore.Catalogo.Domain
7  {
8      public interface IProdutoRepository : IRepository<Produto>
9      {
10         Task<IEnumerable<Produto>> ObterTodos();
11         Task<Produto> ObterPorId(Guid id);
12         Task<IEnumerable<Produto>> ObterPorCategoria(int codigo);
13         Task<IEnumerable<Categoria>> ObterCategorias();
14
15         void Adicionar(Produto produto);
16         void Atualizar(Produto produto);
17
18         void Adicionar(Categoria categoria);
19         void Atualizar(Categoria categoria);
20     }
21 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Threading.Tasks;
4  using NerdStore.Core.Data;
5
6  namespace NerdStore.Vendas.Domain
7  {
8      public interface IPedidoRepository : IRepository<Pedido>
9      {
10         Task<Pedido> ObterPorId(Guid id);
11         Task<IEnumerable<Pedido>> ObterListaPorClienteId(Guid clienteId);
12         Task<Pedido> ObterPedidoRascunhoPorClienteId(Guid clienteId);
13
14         void Adicionar(Pedido pedido);
15         void Atualizar(Pedido pedido);
16
17         Task<PedidoItem> ObterItemPorId(Guid id);
18         Task<PedidoItem> ObterItemPorPedido(Guid pedidoId, Guid produtoId);
19
20         void AdicionarItem(PedidoItem pedidoItem);
21         void AtualizarItem(PedidoItem pedidoItem);
22         void RemoverItem(PedidoItem pedidoItem);
23
24         Task<Voucher> ObterVoucherPorCodigo(string codigo);
25     }
26 }
```

Projeto de Bloco: Engenharia de Software e Modelagem

API Collections – Contexto Catálogo e Vendas

```
1  using System.Collections.Generic;
2  using NerdStore.Core.DomainObjects;
3
4  namespace NerdStore.Catalogo.Domain
5  {
6      // referências
7      public class Categoria : Entity
8      {
9          // referências
10         public string Nome { get; private set; }
11         // referências
12         public int Código { get; private set; }
13
14         // EF Relation
15         // referências
16         public ICollection<Produto> Produtos { get; set; }
17
18         // referências
19         protected Categoria() { }
20
21         // referências
22         public Categoria(string nome, int código)
23         {
24             Nome = nome;
25             Código = código;
26
27             Validar();
28
29             // referências
30             public override string ToString()
31             {
32                 return $"{Nome} - {Código}";
33             }
34
35             // referências
36             public void Validar()
37             {
38                 Validações.ValidarSeVazio(Nome, "O campo Nome da categoria não pode estar vazio");
39                 Validações.ValidarSeIgual(Código, 0, "O campo Código não pode ser 0");
40             }
41         }
42     }
43 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using FluentValidation.Results;
5  using NerdStore.Core.DomainObjects;
6
7  namespace NerdStore.Vendas.Domain
8  {
9      // referências
10     public class Pedido : Entity, IAggregateRoot
11     {
12         // referências
13         public int Código { get; private set; }
14         // referências
15         public Guid ClienteId { get; private set; }
16         // referências
17         public Guid? VoucherId { get; private set; }
18         // referências
19         public bool VoucherUtilizado { get; private set; }
20         // referências
21         public decimal Desconto { get; private set; }
22         // referências
23         public decimal ValorTotal { get; private set; }
24         // referências
25         public DateTime DataCadastro { get; private set; }
26         // referências
27         public PedidoStatus PedidoStatus { get; private set; }
28
29         // referências
30         private readonly List<PedidoItem> _pedidoItems;
31
32         // referências
33         public IReadOnlyCollection<PedidoItem> PedidoItems => _pedidoItems;
34
35         // EF Rel.
36         // referências
37         public Voucher Voucher { get; private set; }
38     }
39 }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Expressões Lambda – Contexto Catálogo e Vendas

```
10  {
11      – referências
12      public class ProdutoRepository : IProdutoRepository
13      {
14          private readonly CatalogoContext _context;
15
16          – referências
17          public ProdutoRepository(CatalogoContext context)
18          {
19              _context = context;
20          }
21
22          – referências
23          public IUnitOfWork UnitOfWork => _context;
24
25
26          – referências
27          public async Task<Produto> ObterPorId(Guid id)
28          {
29              //return await _context.Produtos.AsNoTracking().FirstOrDefaultAsync(p
30              return await _context.Produtos.FindAsync(id);
31
32
33          – referências
34          public async Task<IEnumerable<Produto>> ObterPorCategoria(int codigo)
35          {
36              return await _context
37                  .Produtos
38                  .AsNoTracking()
39                  .Include(p => p.Categoria) ✓
40                  .Where(c => c.Categoria.Codigo == codigo) ✓
41                  .ToListAsync();
42
43      }
44
45  }
```

```
11  {
12      – referências
13      public class PedidoRepository : IPedidoRepository
14      {
15          private readonly VendasContext _context;
16
17          – referências
18          public PedidoRepository(VendasContext context)
19          {
20              _context = context;
21          }
22
23
24          – referências
25          public async Task<Pedido> ObterPorId(Guid id)
26          {
27              return await _context.Pedidos.FindAsync(id);
28
29
30          – referências
31          public async Task<IEnumerable<Pedido>> ObterListaPorClienteId(Guid clienteId)
32          {
33              return await _context
34                  .Pedidos
35                  .AsNoTracking()
36                  .Where(p => p.ClienteId == clienteId) ✓
37                  .ToListAsync();
38
39
40
41          – referências
42          public async Task<Pedido> ObterPedidoRascunhoPorClienteId(Guid clienteId)
43          {
44              var pedido = await _context
45                  .Pedidos
46                  .FirstOrDefaultAsync(p =>
47                      p.ClienteId == clienteId && p.PedidoStatus == PedidoStatus.Rascunho);
48
49              if (pedido == null) return null;
50
51      }
52
53  }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Validações – Contexto Catálogo

```
4  namespace NerdStore.Catalogo.Domain
5  {
6      - referências
7      public class Produto : Entity, IAggregateRoot
8      {
9          - referências
10         public Guid CategoriaId { get; private set; }
11         - referências
12         public string Nome { get; private set; }
13         - referências
14         public string Descricao { get; private set; }
15         - referências
16         public bool Ativo { get; private set; }
17         - referências
18         public decimal Valor { get; private set; }
19         - referências
20         public DateTime DataCadastro { get; private set; }
21         - referências
22         public string Imagem { get; private set; }
23         - referências
24         public int QuantidadeEstoque { get; private set; }
25         - referências
26         public Dimensoes Dimensoes { get; private set; }
27         - referências
28         public Categoria Categoria { get; private set; }
29
30         - referências
31         protected Produto() { }
32         - referências
33         public Produto(string nome,
34                         string descricao,
35                         bool ativo,
36                         decimal valor,
37                         Guid categoriaId,
38                         DateTime dataCadastro,
39                         string imagem,
40                         Dimensoes dimensoes)
41         {
42             CategoriaId = categoriaId;
43             Nome = nome;
44             Descricao = descricao;
45             Ativo = ativo;
46             Valor = valor;
47             DataCadastro = dataCadastro;
48             Imagem = imagem;
49             Dimensoes = dimensoes;
50
51             Validar();
52         }
53     }
54 }
```

```
51     public void AlterarDescricao(string descricao)
52     {
53         Validações.ValidarSeVazio(descricao, "O campo Descricao do produto não pode estar vazio");
54         Descricao = descricao;
55     }
56
57     - referências
58     public void DebitarEstoque(int quantidade)
59     {
60         if (quantidade < 0) quantidade *= -1;
61         if (!PossuiEstoque(quantidade)) throw new DomainException("Estoque insuficiente");
62         QuantidadeEstoque -= quantidade;
63     }
64
65     - referências
66     public void ReporEstoque(int quantidade)
67     {
68         QuantidadeEstoque += quantidade;
69     }
70
71     - referências
72     public bool PossuiEstoque(int quantidade)
73     {
74         return QuantidadeEstoque >= quantidade;
75     }
76
77     - referências
78     public void Validar()
79     {
80         - Validações.ValidarSeVazio(Nome, "O campo Nome do produto não pode estar vazio");
81         - Validações.ValidarSeVazio(Descricao, "O campo Descricao do produto não pode estar vazio");
82         - Validações.ValidarSeIgual(CategoriaId, Guid.Empty, "O campo CategoriaId do produto não pode estar vazio");
83         - Validações.ValidarSeMenorQue(Valor, 1, "O campo Valor do produto não pode ser menor igual a 0");
84         - Validações.ValidarSeVazio(Imagen, "O campo Imagem do produto não pode estar vazio");
85     }
86 }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Validações – Contexto Vendas

```
1  using System;
2  using FluentValidation;
3  using NerdStore.Core.Messages;
4
5  namespace NerdStore.Vendas.Application.Commands
6  {
7      public class AdicionarItemPedidoCommand : Command
8      {
9          public Guid ClienteId { get; private set; }
10         public Guid ProdutoId { get; private set; }
11         public string Nome { get; private set; }
12         public int Quantidade { get; private set; }
13         public decimal ValorUnitario { get; private set; }
14
15         public AdicionarItemPedidoCommand(Guid clienteId,
16                                           Guid produtoId,
17                                           string nome,
18                                           int quantidade,
19                                           decimal valorUnitario)
20         {
21             ClienteId = clienteId;
22             ProdutoId = produtoId;
23             Nome = nome;
24             Quantidade = quantidade;
25             ValorUnitario = valorUnitario;
26         }
27
28         public override bool EhValido()
29         {
30             ValidationResult = new AdicionarItemPedidoValidation().Validate(this);
31             return ValidationResult.IsValid;
32         }
33     }
```

```
35  public class AdicionarItemPedidoValidation : AbstractValidator<AdicionarItemPedidoCommand>
36  {
37      public AdicionarItemPedidoValidation()
38      {
39          RuleFor(c => c.ClienteId)
40              .NotEqual(Guid.Empty)
41              .WithMessage("Id do cliente inválido");
42
43          RuleFor(c => c.ProdutoId)
44              .NotEqual(Guid.Empty)
45              .WithMessage("Id do produto inválido");
46
47          RuleFor(c => c.Nome)
48              .NotEmpty()
49              .WithMessage("O nome do produto não foi informado");
50
51          RuleFor(c => c.Quantidade)
52              .GreaterThanOrEqualTo(0)
53              .WithMessage("A quantidade mínima de um item é 1");
54
55          RuleFor(c => c.Quantidade)
56              .LessThan(15)
57              .WithMessage("A quantidade máxima de um item é 15");
58
59          RuleFor(c => c.ValorUnitario)
60              .GreaterThanOrEqualTo(0)
61              .WithMessage("O valor do item precisa ser maior que 0");
62
63      }
64  }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Exceções – Contexto Vendas

```
1  using System.Text.RegularExpressions;
2
3  namespace NerdStore.Core.DomainObjects
4  {
5      11 referências
6      public class Validações
7      {
8          2 referências
9          public static void ValidarSeIgual(object object1, object object2, string mensagem)
10         {
11             if (object1.Equals(object2))
12             {
13                 throw new DomainException(mensagem);
14             }
15         }
16
17         0 referências
18         public static void ValidarSeDiferente(object object1, object object2, string mensagem)
19         {
20             if (!object1.Equals(object2))
21             {
22                 throw new DomainException(mensagem);
23             }
24         }
25
26         0 referências
27         public static void ValidarSeDiferente(string pattern, string valor, string mensagem)
28         {
29             var regex = new Regex(pattern);
30
31             if (!regex.IsMatch(valor))
32             {
33                 throw new DomainException(mensagem);
34             }
35         }
36
37         0 referências
38         public static void ValidarTamanho(string valor, int maximo, string mensagem)
39         {
40             var length = valor.Trim().Length;
41             if (length > maximo)
42             {
43                 throw new DomainException(mensagem);
44             }
45         }
46     }
47 }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Tratamento de Erros – Contexto Vendas

```
9  namespace NerdStore.WebApp.MVC.Extensions
10 {
11     2 referências
12     public class ExceptionMiddleware
13     {
14         private readonly RequestDelegate _next;
15
16         0 referências
17         public ExceptionMiddleware(RequestDelegate next)
18         {
19             _next = next;
20         }
21
22         0 referências
23         public async Task InvokeAsync(HttpContext httpContext)
24         {
25
26             try
27             {
28                 await _next(httpContext);
29             }
30             catch (CustomHttpRequestException ex)
31             {
32                 HandleRequestExceptionAsync(httpContext, ex.StatusCode);
33             }
34             catch (ValidationApiException ex)
35             {
36                 HandleRequestExceptionAsync(httpContext, ex.StatusCode);
37             }
38             catch (ApiException ex)
39             {
40                 HandleRequestExceptionAsync(httpContext, ex.StatusCode);
41             }
42             catch (BrokenCircuitException)
43             {
44                 HandleCircuitBreakerExceptionAsync(httpContext);
45             }
46         }
47     }
48 }
```

```
44
45
46     catch (RpcException ex)
47     {
48         //400 Bad Request      INTERNAL
49         //401 Unauthorized   UNAUTHENTICATED
50         //403 Forbidden       PERMISSION_DENIED
51         //404 Not Found      UNIMPLEMENTED
52
53         var statusCode = ex.StatusCode switch
54         {
55             StatusCode.Internal => 400,
56             StatusCode.Unauthenticated => 401,
57             StatusCode.PermissionDenied => 403,
58             StatusCode.Unimplemented => 404,
59             _ => 500
60         };
61
62         var httpStatusCode = ( HttpStatusCode )Enum.Parse( typeof( HttpStatusCode ), statusCode.ToString() );
63         HandleRequestExceptionAsync( httpContext, httpStatusCode );
64     }
65     catch ( HttpRequestException ex )
66     {
67         var statusCode = ex.StatusCode switch
68         {
69             HttpStatusCode.BadRequest => 400,
70             HttpStatusCode.Unauthorized => 401,
71             HttpStatusCode.Forbidden => 403,
72             HttpStatusCode.NotFound => 404,
73             _ => 500
74         };
75
76         var httpStatusCode = ( HttpStatusCode )Enum.Parse( typeof( HttpStatusCode ), statusCode.ToString() );
77         HandleRequestExceptionAsync( httpContext, httpStatusCode );
78     }
79 }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Tratamento de Erros – Contexto Vendas

```
0 referencias
72     public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
73     {
74         app.UseForwardedHeaders();
75
76         //if (env.IsDevelopment())
77         //{
78             //    app.UseDeveloperExceptionPage();
79             //    //app.UseDatabaseErrorPage();
80         //}
81         //else
82         //{
83             //    app.UseExceptionHandler("/Home/Error");
84             //    // The default HSTS value is 30 days. You may want to change this for production scenarios,
85             //    app.UseHsts();
86         //}
87
88         app.UseExceptionHandler("/erro/500");
89         app.UseStatusCodePagesWithRedirects("/erro/{0}");
90         app.UseHsts();
91         app.UseHttpsRedirection(); ✓
92         app.UseStaticFiles();
93         app.UseRouting();
94         app.UseCookiePolicy();
95         app.UseAuthentication();
96         app.UseMiddleware<ExceptionMiddleware>(); ✓
97
98         app.UseEndpoints(endpoints =>
99         {
100             endpoints.MapControllerRoute(
101                 name: "default",
102                 pattern: "{controller=Vitrine}/{action=Index}/{id?}");
103
104             endpoints.MapRazorPages();
105
106         });
107
108     }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Testes de Unidade – Contexto Catálogo

```
4  namespace NerdStore.Catalogo.Domain.Tests
5  {
6      // referências
7      public class ProdutoTests
8      {
9          [Fact]
10         // referências
11         public void Produto_ValidacoesDevemRetornarExceptions()
12         {
13             // Arrange & Act & Assert
14
15             var ex = Assert.Throws<DomainException>(() =>
16                 new Produto(string.Empty, "Descricao", false, 100, Guid.NewGuid(), DateTime.Now, "Imagen", new Dimensoes(1, 1, 1))
17             );
18
19             Assert.Equal("O campo Nome do produto não pode estar vazio", ex.Message);
20
21             ex = Assert.Throws<DomainException>(() =>
22                 new Produto("Nome", string.Empty, false, 100, Guid.NewGuid(), DateTime.Now, "Imagen", new Dimensoes(1, 1, 1))
23             );
24
25             Assert.Equal("O campo Descricao do produto não pode estar vazio", ex.Message);
26
27             ex = Assert.Throws<DomainException>(() =>
28                 new Produto("Nome", "Descricao", false, 0, Guid.NewGuid(), DateTime.Now, "Imagen", new Dimensoes(1, 1, 1))
29             );
30
31             Assert.Equal("O campo Valor do produto não pode se menor igual a 0", ex.Message);
32
33             ex = Assert.Throws<DomainException>(() =>
34                 new Produto("Nome", "Descricao", false, 100, Guid.Empty, DateTime.Now, "Imagen", new Dimensoes(1, 1, 1))
35             );
36
37             Assert.Equal("O campo Categoriald do produto não pode estar vazio", ex.Message);
38
39             ex = Assert.Throws<DomainException>(() =>
40                 new Produto("Nome", "Descricao", false, 100, Guid.NewGuid(), DateTime.Now, string.Empty, new Dimensoes(1, 1, 1))
41             );
42
43             Assert.Equal("O campo Imagem do produto não pode estar vazio", ex.Message);
44
45             ex = Assert.Throws<DomainException>(() =>
46                 new Produto("Nome", "Descricao", false, 100, Guid.NewGuid(), DateTime.Now, "Imagen", new Dimensoes(0, 1, 1))
47             );
48
49             Assert.Equal("O campo Altura não pode ser menor ou igual a 0", ex.Message);
50
51         }
52     }
```

Projeto de Bloco: Engenharia de Software e Modelagem

CRUD Classes de Domínio – Contexto Catálogo

```
1  using System;
2  using System.Collections.Generic;
3  using System.Threading.Tasks;
4  using NerdStore.Core.Data;
5
6  namespace NerdStore.Catalogo.Domain
7  {
8      public interface IProdutoRepository : IRepository<Produto>
9      {
10          Task<IEnumerable<Produto>> ObterTodos();
11          Task<Produto> ObterPorId(Guid id);
12          Task<IEnumerable<Produto>> ObterPorCategoria(int codigo);
13          Task<IEnumerable<Categoria>> ObterCategorias();
14
15          void Adicionar(Produto produto);
16          void Atualizar(Produto produto);
17
18          void Adicionar(Categoria categoria);
19          void Atualizar(Categoria categoria);
20      }
21  }
```

Projeto de Bloco: Engenharia de Software e Modelagem

CRUD Classes de Domínio – Contexto Vendas / Pagamento

```
1  using System;
2  using System.Collections.Generic;
3  using System.Threading.Tasks;
4  using NerdStore.Core.Data;
5
6  namespace NerdStore.Vendas.Domain
7  {
8      // referências
9      public interface IPedidoRepository : IRepository<Pedido>
10     {
11         // referências
12         Task<Pedido> ObterPorId(Guid id);
13         // referências
14         Task<IEnumerable<Pedido>> ObterListaPorClienteId(Guid clienteId);
15         // referências
16         Task<Pedido> ObterPedidoRascunhoPorClienteId(Guid clienteId);
17         // referências
18         void Adicionar(Pedido pedido);
19         // referências
20         void Atualizar(Pedido pedido);
21
22         // referências
23         Task<PedidoItem> ObterItemPorId(Guid id);
24         // referências
25         Task<PedidoItem> ObterItemPorPedido(Guid pedidoId, Guid produtoId);
26         // referências
27         void AdicionarItem(PedidoItem pedidoItem);
28         // referências
29         void AtualizarItem(PedidoItem pedidoItem);
30         // referências
31         void RemoverItem(PedidoItem pedidoItem);
32
33         // referências
34         Task<Voucher> ObterVoucherPorCodigo(string codigo);
35     }
36 }
```

```
1  using NerdStore.Core.Data;
2
3  namespace NerdStore.Pagamentos.Business
4  {
5      // referências
6      public interface IPagamentoRepository : IRepository<Pagamento>
7      {
8          // referências
9          void Adicionar(Pagamento pagamento);
10         // referências
11         void AdicionarTransacao(Transacao transacao);
12     }
13 }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Consultar e Manipular usando LINQ

Projeto de Bloco: Engenharia de Software e Modelagem

Mapeamento EF Core – Contexto Catálogo

```
1  using Microsoft.EntityFrameworkCore;
2  using Microsoft.EntityFrameworkCore.Metadata.Builders;
3  using NerdStore.Catalogo.Domain;
4
5  namespace NerdStore.Catalogo.Data.Mappings
6  {
7      // referências
8      public class ProdutoMapping : IEntityTypeConfiguration<Produto>
9      {
10         // referências
11         public void Configure(EntityTypeBuilder<Produto> builder)
12         {
13             builder.HasKey(c => c.Id);
14
15             builder.Property(c => c.Nome)
16                 .IsRequired()
17                 .HasColumnType("varchar(250)");
18
19             builder.Property(c => c.Descricao)
20                 .IsRequired()
21                 .HasColumnType("varchar(500)");
22
23             builder.Property(c => c.Imagem)
24                 .IsRequired()
25                 .HasColumnType("varchar(250)");
26
27             builder.OwnsOne(c => c.Dimensoes, cm =>
28             {
29                 cm.Property(c => c.Altura)
30                     .HasColumnName("Altura")
31                     .HasColumnType("int");
32
33                 cm.Property(c => c.Largura)
34                     .HasColumnName("Largura")
35                     .HasColumnType("int");
36
37                 cm.Property(c => c.Profundidade)
38                     .HasColumnName("Profundidade")
39                     .HasColumnType("int");
40             });
41
42             builder.ToTable("Produtos");
43         }
44     }
```

```
1  using Microsoft.EntityFrameworkCore;
2  using Microsoft.EntityFrameworkCore.Metadata.Builders;
3  using NerdStore.Catalogo.Domain;
4
5  namespace NerdStore.Catalogo.Data.Mappings
6  {
7      // referências
8      public class CategoriaMapping : IEntityTypeConfiguration<Categoria>
9      {
10         // referências
11         public void Configure(EntityTypeBuilder<Categoria> builder)
12         {
13             builder.HasKey(c => c.Id);
14
15             builder.Property(c => c.Nome)
16                 .IsRequired()
17                 .HasColumnType("varchar(250)");
18
19             // 1 : N => Categorias : Produtos
20             builder.HasMany(c => c.Produtos)
21                 .WithOne(p => p.Categoria)
22                 .HasForeignKey(p => p.CategoriaId);
23
24         }
25     }
26 }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Mapeamento EF Core – Contexto Vendas

```
1  using Microsoft.EntityFrameworkCore;
2  using Microsoft.EntityFrameworkCore.Metadata.Builders;
3  using NerdStore.Vendas.Domain;
4
5  namespace NerdStore.Catalogo.Data.Mappings
6  {
7      // referências
8      public class PedidoMapping : IEntityTypeConfiguration<Pedido>
9      {
10         // referências
11         public void Configure(EntityTypeBuilder<Pedido> builder)
12         {
13             builder.HasKey(c => c.Id);
14
15             builder.Property(c => c.Codigo)
16                 .HasDefaultValueSql("NEXT VALUE FOR MinhaSequencia");
17
18             // 1 : N => Pedido : PedidoItems
19             builder.HasMany(c => c.PedidoItems)
20                 .WithOne(c => c.Pedido)
21                 .HasForeignKey(c => c.PedidoId);
22
23             builder.ToTable("Pedidos");
24         }
25     }
}
```

```
1  using Microsoft.EntityFrameworkCore;
2  using Microsoft.EntityFrameworkCore.Metadata.Builders;
3  using NerdStore.Vendas.Domain;
4
5  namespace NerdStore.Catalogo.Data.Mappings
6  {
7      // referências
8      public class PedidoItemMapping : IEntityTypeConfiguration<PedidoItem>
9      {
10         // referências
11         public void Configure(EntityTypeBuilder<PedidoItem> builder)
12         {
13             builder.HasKey(c => c.Id);
14
15             builder.Property(c => c.ProdutoNome)
16                 .IsRequired()
17                 .HasColumnType("varchar(250)");
18
19             // 1 : N => Pedido : Pagamento
20             builderhasOne(c => c.Pedido)
21                 .WithMany(c => c.PedidoItems);
22
23             builder.ToTable("PedidoItems");
24         }
25     }
}
```

Projeto de Bloco: Engenharia de Software e Modelagem

Desacoplar Camada de Persistência com MVC

```
1  using System;
2  using System.Threading.Tasks;
3  using Microsoft.AspNetCore.Mvc;
4  using NerdStore.Catalogo.Application.Services;
5
6  namespace NerdStore.WebApp.MVC.Controllers
7  {
8      – referências
9      public class VitrineController : Controller
10     {
11         private readonly IProdutoAppService _produtoAppService;
12
13         – referências
14         public VitrineController(IProdutoAppService produtoAppService)
15         {
16             _produtoAppService = produtoAppService;
17
18             [HttpGet]
19             [Route("")]
20             [Route("vitrine")]
21             – referências
22             public async Task<IActionResult> Index()
23             {
24                 return View(await _produtoAppService.ObterTodos());
25             }
26
27             [HttpGet]
28             [Route("produto-detalhe/{id}")]
29             – referências
30             public async Task<IActionResult> ProdutoDetalhe(Guid id)
31             {
32                 return View(await _produtoAppService.ObterPorId(id));
33             }
34         }
35     }
36 }
```

```
1  using System.Threading.Tasks;
2  using MediatR;
3  using Microsoft.AspNetCore.Mvc;
4  using NerdStore.Core.Communication.Mediator;
5  using NerdStore.Core.Messages.CommonMessages.Notifications;
6  using NerdStore.Vendas.Application.Queries;
7
8  namespace NerdStore.WebApp.MVC.Controllers
9  {
10     – referências
11     public class PedidoController : ControllerBase
12     {
13         private readonly IPedidoQueries _pedidoQueries;
14
15         – referências
16         public PedidoController(IPedidoQueries pedidoQueries,
17                               INotificationHandler<DomainNotification> notifications,
18                               IMediatorHandler mediatorHandler) : base(notifications, mediatorHandler)
19         {
20             _pedidoQueries = pedidoQueries;
21         }
22
23         [Route("meus-pedidos")]
24         – referências
25         public async Task<IActionResult> Index()
26         {
27             return View(await _pedidoQueries.ObterPedidosCliente(ClienteId));
28         }
29     }
30 }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Desacoplar Camada de Persistência com Web API

Projeto de Bloco: Engenharia de Software e Modelagem

Documentação com Swagger/OpenAPI

Projeto de Bloco: Engenharia de Software e Modelagem

Requisitos de Alto Nível - RANs

Projeto de Bloco: Engenharia de Software e Modelagem



Os RANs são descrições do que o sistema/produto deve ou não fazer. Eles não definem como deve ser feito, pois isto já compete a área técnica. Eles geralmente se concentram nos resultados desejados e nas funcionalidades gerais que o sistema ou produto deve ter para atender às necessidades dos usuários e das partes interessadas.

Projeto de Bloco: Engenharia de Software e Modelagem



Projeto de Bloco: Engenharia de Software e Modelagem

1. O sistema deve permitir que os usuários façam login e gerenciem suas contas.
2. O produto deve ser fácil de usar para usuários com habilidades limitadas em tecnologia.
3. O sistema deve ser capaz de processar um grande volume de transações em tempo real.
4. O produto deve ser seguro e proteger os dados confidenciais dos usuários.
5. O sistema deve ser escalável e capaz de lidar com o crescimento futuro da empresa.
6. O Produto enviado deverá ser rastreável.

Projeto de Bloco: Engenharia de Software e Modelagem

Requisitos Funcionais

Projeto de Bloco: Engenharia de Software e Modelagem

Requisitos Funcionais		Requisito	Regra de negócio
[RF-001]	Manter Produto	RF 01	QUANTIDADE: produto é considerado disponível quando está ativo e tem estoque > 0
[RF-002]	Manter Estoque	RF 01	PRODUTO: tem que ter obrigatoriamente id, nome, descrição, valor e imagem
[RF-003]	Manter Categoria	RF 01	PRODUTO: dimensões devem ter altura, largura e profundidade maior ou igual a zero
[RF-004]	Gerir venda	RF 01	PRODUTO: a formatação das dimensões deve ser "L x A x P"
[RF-005]	Aplicar voucher pedido	RF 01	PRODUTO: Sistema deve exibir detalhes de produto
[RF-006]	Estornar estoque	RF 02	ESTOQUE: Ao concluir uma compra, sistema deve debituar quantidade de itens de estoque
		RF 03	PRODUTO: Sistema deve listar todos os produtos disponíveis
		RF 03	PRODUTO: Sistema deve listar produtos disponíveis filtrados por categoria
		RF 03	CATEGORIA: deve ter obrigatoriamente Nome e código
		RF 04	PAGAMENTO: Pagamento pode ser realizado por cartão de crédito e boleto
		RF 04	CARRINHO: Precisa ter 1 ou mais itens para ser fechado
		RF 04	PAGAMENTO: Status de transação pode ser Pago ou Recusado
		RF 04	COMPRA: Sistema deve recuperar resumo de compra desejado
		RF 05	VOUCHER: Carrinho de compras - Desconto absoluto e/ou percentual.
		RF 06	PAGAMENTO: Se o pagamento não for realizado com sucesso, a compra não deve ser concluída
		RF 06	ESTORNO: Em casos de devolução de produto, estoque deve ser acrescido da quantidade devolvida

Projeto de Bloco: Engenharia de Software e Modelagem

Requisitos Não Funcionais

Projeto de Bloco: Engenharia de Software e Modelagem

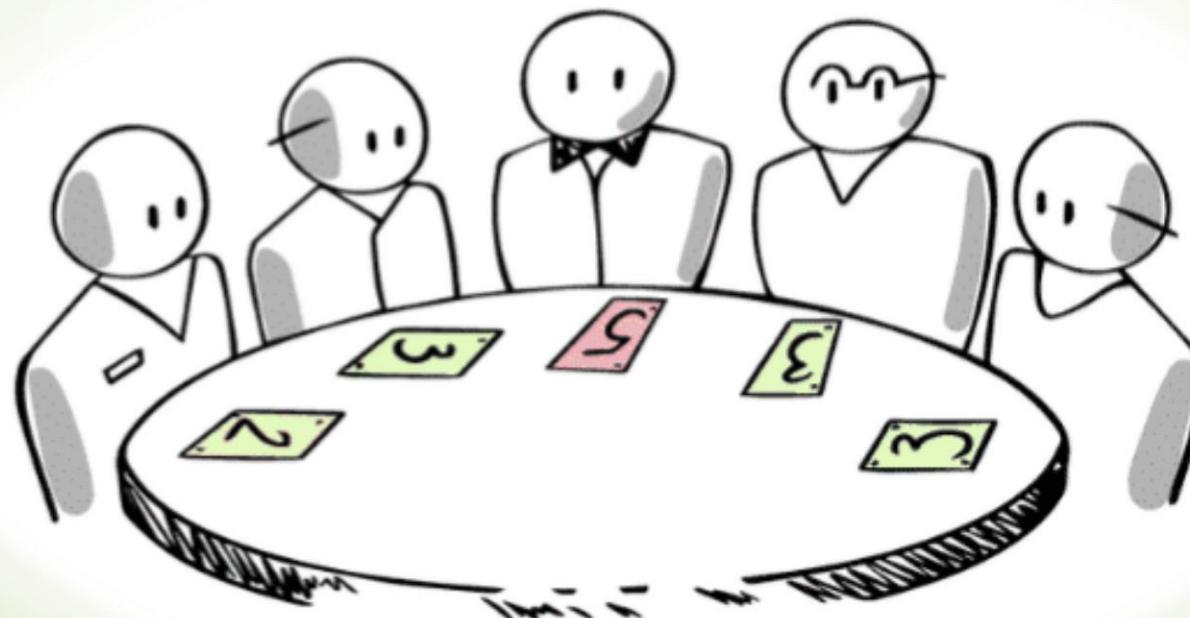
Código	Requisitos Não Funcionais	Indicadores
RNF 01	Desempenho	Tempo de Resposta das requisições(1 a cada 100 ms)
RNF 02	Disponibilidade	Funcional em tempo integral (24h/7 dias por semana)
RNF 03	Escalabilidade	picos de consumo de até 1k acessos/min
RNF 04	Segurança	Autenticação JWT / criptografia Char 256
RNF 05	Privacidade	Cumprimento da LGPD

Projeto de Bloco: Engenharia de Software e Modelagem

Matriz de Rastreabilidade

Projeto de Bloco: Engenharia de Software e Modelagem

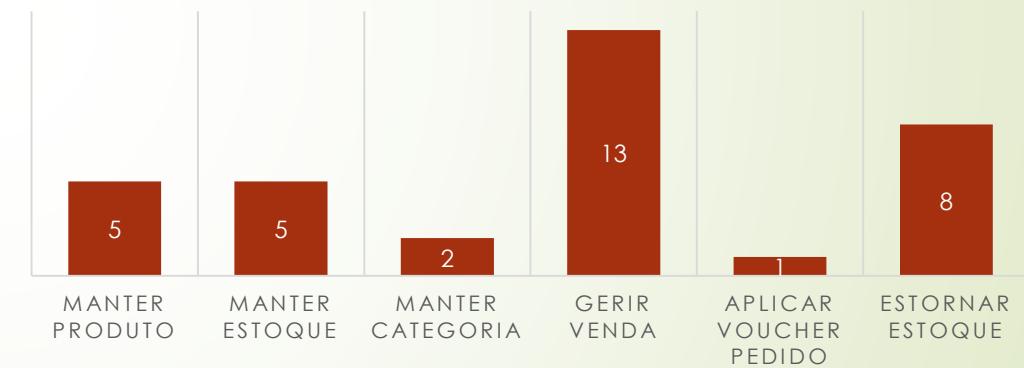
Processo de estimativa



Projeto de Bloco: Engenharia de Software e Modelagem

código	Requisito Funcional	Complexidade Estimada
[RF-001]	Manter Produto	5
[RF-002]	Manter Estoque	5
[RF-003]	Manter Categoria	2
[RF-004]	Gerir venda	13
[RF-005]	Aplicar voucher pedido	1
[RF-006]	Estornar estoque	8

COMPLEXIDADE ESTIMADA



Projeto de Bloco: Engenharia de Software e Modelagem

Cronograma de implementação

Requisitos	Semanas							
	1	2	3	4	5	6	7	8
Manter Produto								
Manter Estoque								
Manter Categoria								
Gerir venda								
Aplicar voucher pedido								
Estornar estoque								

Cronograma de lançamento

Requisitos	Releases			
	1	2	3	Final
Manter Produto				
Manter Estoque				
Manter Categoria				
Gerir venda				
Aplicar voucher pedido				
Estornar estoque				

Projeto de Bloco: Engenharia de Software e Modelagem

Proposta de implantação



- AKS (Serviço de Kubernetes do Azure)
- Registro de containers do Azure
- Aplicativo Web para Contêiners
- Banco de dados SQL do Azure
- Cache do Redis para Azure
- Armazenamento Blobs do Azure
- Key Vault
- API Gateway
- Firewall

Projeto de Bloco: Engenharia de Software e Modelagem

Cronograma geral

Fase do projeto	Meses					
	1	2	3	4	5	..
Implementação						
Implantação						
Mantenimento						
Sustentação						

Projeto de Bloco: Engenharia de Software e Modelagem

Custo do software

Manutenção	R\$ 3.750,00
Implantação	R\$ 6.000,00
Implementação	R\$ 15.454,55
Gerenciamento	20%
Margem de lucro	40%
CUSTO TOTAL	R\$ 30.246,86
Manutenção extra	R\$ 1.250,00 / mês

Projeto de Bloco: Engenharia de Software e Modelagem

► Referências

MODELAGEM de Domínios Ricos. desenvolvedor.io/. Disponível em: <https://desenvolvedor.io/curso/modelagem-de-dominios-ricos> . Acesso em: 02 jun. 2022.

MODELAGEM de Software. lms.infnet.edu.br. Disponível em: <https://lms.infnet.edu.br/moodle/course/view.php?id=5928> . Acesso em: 10 fev. 2022.

ENGENHARIA de Software Aplicada. lms.infnet.edu.br. Disponível em: <https://lms.infnet.edu.br/moodle/course/view.php?id=5927> . Acesso em: 11 dez. 2022.

MODELAGEM de Dados UML. udemy.com. Disponível em: <https://www.udemy.com/course/uml-diagrama-de-classes/learn/lecture/9881660?start=255#overview> . Acesso em: 17 abr. 2022.